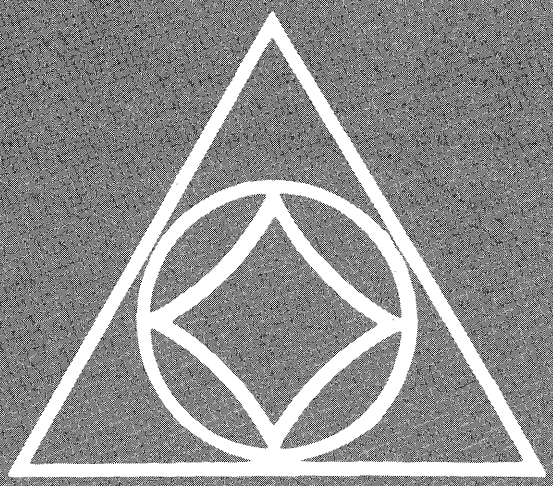


AFIPS

CONFERENCE PROCEEDINGS

SAKTI P. GHOSH
and
LEONARD Y. LIU
Editors and Program Co-Chairmen
STEPHEN W. MILLER
Conference Chairman

1978

NATIONAL COMPUTER CONFERENCE

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

June 5-8, 1978
Anaheim, California

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

PART I—APPLICATIONS

CONTENTS

IMAGE PROCESSING FOR REMOTE SENSING

EFTS—ELECTRONIC FUND TRANSFER SYSTEM

HOME AND HOBBYING COMPUTING

HOME AND HOBBYING COMPUTING—THE CURRENT SITUATION AND THE FORESEEABLE FUTURE

Chairperson—Jim C. Warren, Jr.

PERSONAL HARDWARE, SOFTWARE, AND APPLICATIONS

Chairperson—Carl T. Helmers, Jr.

SIMULATION

MICROPROGRAMMING AND SIMULATION

Chairperson—Gary J. Nutt

PERIPHERAL AND MULTI-PROCESSORS IN SIMULATION

Chairperson—Walter J. Karplus

NEW APPLICATIONS

P. E. Mantey—Area Director

APPLICATIONS OF MINICOMPUTERS TO LIBRARY SYSTEMS AND SERVICES

Chairperson—Laura J. Rainey

PART II—METHODOLOGY

PERFORMANCE MEASUREMENT AND EVALUATION

DP MANAGEMENT AND ADMINISTRATION

SOFTWARE DESIGN AND ANALYSIS

Chairperson—Donna Dunaway

PROGRAMMING METHODOLOGY

Chairperson—Susan Gerhart

SOFTWARE VERIFICATION, VALIDATION, AND TESTING TECHNIQUES

Chairperson—Edward Miller

USER EXPERIENCE WITH NEW SOFTWARE METHODS

Chairperson—Victor R. Basili

PART III—SYSTEMS

DATA NETWORKS

DISTRIBUTED SYSTEMS

David J. Farber—Area Director

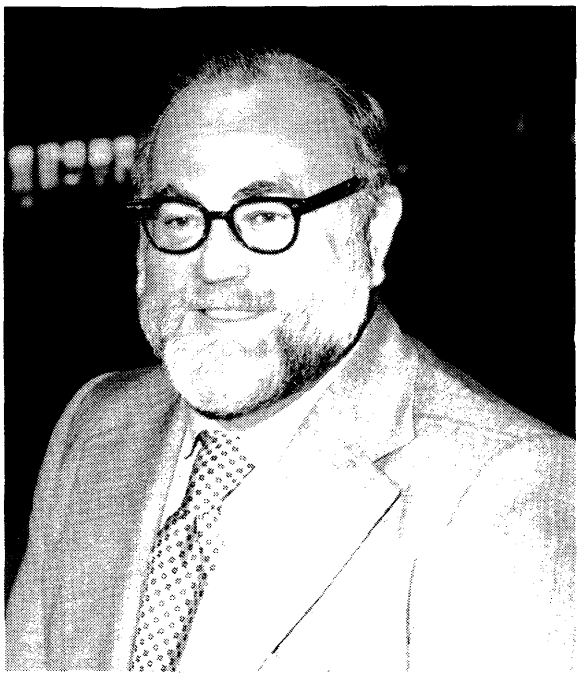
DESIGN METHODOLOGY FOR DISTRIBUTED DATA PROCESSING

Chairperson—E. Douglas Jensen

SOFTWARE, APPLICATIONS AND PROGRAMMING LANGUAGES FOR DISTRIBUTED ENVIRONMENT

Chairperson—Eric Manning

PART IV—PEOPLE AND SOCIETY



STEPHEN W. MILLER
Conference Chairman
SRI International
Menlo Park, California

Preface

We are pleased you have chosen to enjoy the 1978 National Computer Conference (NCC). Frequently the NCC is judged to be "too large" to have a theme. This year we bring you two themes: the primary theme is the national energy problem and how computers can contribute to its alleviation; the secondary theme is the growth of the computer industry, especially its fastest growing segment, personal computing.

The energy and computation theme is highlighted in the keynote address and in a powerful panel for the Tuesday evening's Symposium on Developing Energy and Computing Technology. It is continued by the luncheon speakers and by more than 10 percent of the Technical and Professional Program.

On coping with growth in the computer industry, a portion of the Technical and Professional Program deals with personal computing and with developing education curricula. The principal highlight is the Personal Computing Festival, with exhibits, sessions, and contests all its own. This is the first time the National Computer Conference has had separate registration with no age limit on attendance for a portion of its program.

This document forms the principal permanent record of the Technical and Professional Program of this conference. As such, it is a vital part of the documentation and information-dissemination system in our community. The explosive growth of literature in the last decade makes it imperative that this dissemination system be examined carefully. One such examination has been conducted by Thomas J. Allan, in his interesting NSF-funded, ten-year study of this information-dissemination system and reported in *The Management of Technology Flow* (MIT 1977). Most such studies lump scientific and engineering literature; however, he draws a careful distinction between them. One part of that distinction is that a given topic generally appears earlier in time in scientific literature than in engineering literature. During this time delay three very important things occur: *diffusion* of the information through the community; *translation* into the language of current technical practice; and *evaluation* of the concepts. This evaluation tends to be related to enterprise objectives and, hence, of strong interest to technical management.

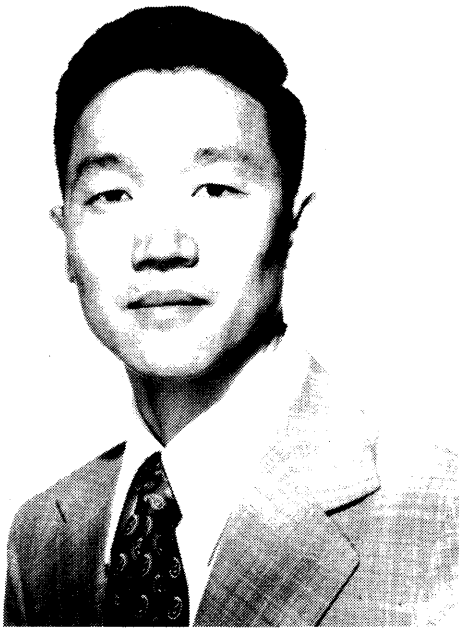
Despite the explosive growth in the number of conferences, symposia, seminars, and workshops dealing with the

computer industry, the National Computer Conference has remained the premier conference to enhance this diffusion, speed this translation, and sharpen this evaluation. NCC is supported by 15 individual technical societies which provide it with the multidisciplinary support that makes it the focal point of excellence in the literature for technology and management.

To this end, in 1978 we have not equated originality with quality. Rather, we have brought the *best judgment* of a very strong technical and professional program committee together to ensure that the topics of importance to the community at large are represented. The best of these topics are presented in either paper or panel form. You will note there are two papers in this proceedings that have been presented earlier for a smaller audience. "The art of artificial intelligence—Themes and case studies of knowledge engineering," by Dr. Edward A. Feigenbaum and "The ubiquity of discovery," by Dr. Douglas B. Lenat. Within the AI community these have been judged "best papers" and hence are repeated here to speed the translation process and to enhance the diffusion throughout the larger information processing community.

Likewise, we have attempted to present both sides of controversial issues so that society itself can participate in the evaluation. Only in a large multidisciplinary forum of this type can we begin to provide approaches to solutions of the very pervasive problems, such as the national energy problem, the facets of a public information policy, the problems of federal regulation, etc., that face us today.

My undying thanks go to the literally hundreds of people whose efforts have gone into bringing you this fine conference, but especially to those who have dedicated their time, talent, and effort to serve on its many committees. The names of most of them are acknowledged in the back of this proceedings.



LEONARD Y. LIU
Program Co-Chairman
IBM Corporation
San Jose, California

SAKTI P. GHOSH
Program Co-Chairman
IBM Corporation
San Jose, California

Introduction

SHAPING THE PROGRAM

Of the many criteria that were followed in creating the technical program committee of 1978, the one that played the dominant role was "excellence." We believed that attracting top people to participate in the technical program committee would result in an excellent technical program. We were indeed fortunate in attracting top technical and professional people from industry, government, universities, and other parts of society which are associated with the computer industry. We also included in our committee top professionals from our sponsoring societies; IEEE Computer Society, ACM, SCS, and DPMA. Cooperation from each of the non-sponsoring constituent societies of AFIPS was requested through their Presidents by our conference vice-chairman.

The program committee in its first meeting in December 1976 established the following philosophy and aims: (i) The number and reactions of attendees are the major criteria for measuring the success of the conference; (ii) Seek broad base appeal—do not concentrate on applications for industrial usage and ignore the system questions. Do not concentrate on the systems architecture problems and ignore the general public; (iii) Orient the program toward industrial and commercial audiences. With this charter in hand, the technical program committee began identifying appropriate technical areas for the program and created a broadly structured program with built-in adaptability to accomodate unsolicited high quality papers.

PROGRAM STRUCTURE

The program is structured into four major areas: Applications, Methodology, Systems and People and Society. Each major area is subdivided into multiple basic areas. Each member of the program committee was responsible for at least one basic area. Each basic area was headed by an area director; some area directors were also a member of the program committee. The area directors were selected for both their professional reputation and motivation. The basic areas cover most of the traditional important aspects of computer knowledge. We have tried to capture some emerging important areas such as Computer Models in Solving World's Energy Problem, Electronic Fund Transfers, Special Purpose Terminals, Office Automation, Home and

Hobbying Computing, Evolution of New Hardware Technology, and Legislation and Its Impact. For the first time in NCC's history, we have asked a sister nation, namely Japan, to organize an area in our conference.

One of the primary responsibilities of the area directors was to solicit high quality papers. The rule "every paper will be refereed" was applied universally to all solicited and unsolicited papers with the exception of two invited papers. The program contains many panel sessions. Each panelist was asked to prepare a position paper and have it approved by the session chairperson. In some cases the position papers of panelists participating in a session were combined into a panel paper and are included in these proceedings. Most of the area directors have included an overview of their area.

We hope you will find these proceedings of the program valuable and rewarding for many years to come.

PART I—APPLICATIONS

Area Director:
William F. Rousseau
Lawrence Livermore Laboratory
Livermore, California

Computer models in solving the world's energy problems

The '73-'74 Oil Embargo and the more recent shortages of natural gas have made the general public energy conscious. Moreover, depleted oil and gas resources, energy conservation legislation, and higher energy prices are creating problems in industry. Significant government programs have been initiated; more, possibly much more, is yet to come. However, causes and cures of our energy problems are controversial. Complexity of energy supply and use play as big a role in this confusion as differences in economic and political philosophy. The complexity makes it difficult to accurately assess the impact of corrective action or even if any is needed and is forcing many policy analysts to turn to computer models for help in making major public and private energy policy decisions. More conventional applications of computers to energy are also expanding with the objectives of finding or recovering more oil and gas or making energy systems more efficient.

These sessions on the applications of computers in solving the worlds energy problems can only sample the many possibilities. Two sessions, "Energy Decision Analysis" and "Energy Modeling Panel," are devoted to the very significant application of computer modeling techniques to energy policy analysis. Two other sessions, "Computers in Petroleum Exploration" and "Computers in Energy Technology," provide examples from the technological and engineering side.

"Energy Decision Analysis," chaired by Robert Karsan, has papers by working policy analysts. Tani's paper reviews an analysis of President Ford's proposed synthetic fuels commercialization program that seemed to have had a major impact—to kill the program. Brock's paper addresses the economic theory used to construct several of the large energy models. Schweizer describes a model developed to predict peak electric load demand—a very important quantity in planning electric power system expansion and energy use. Rice and Meeske describe a model for natural gas demand forecasting based on interfuel competition. Although computers are not emphasized in these papers, the work described would not be practical without them.

"Computers in energy technology" chaired by Julius Chang, provides examples of computer use to improve existing technology. Kamel and Wolf show how computers are being applied to build safer and more energy efficient automobiles. Westbrock and Haselman describe work on modeling what goes on in the cylinder

on an internal combustion engine for computer simulation. Price presents a model which relates to improved oil recovery from existing wells.

"Computers in Petroleum exploration," chaired by Pierre L. Goupillaud, has papers on several aspects of oil and gas exploration. Hoffman and Ternus discuss application of computers to petroleum exploration in general. Savit predicts enormous data processing requirements for geophysical data. Selzler gets into seismic modeling.

"Energy Modeling Panel," chaired by William F. Rousseau, addresses several significant questions raised by the widespread use of energy models in policy analysis. Participants all have wide experience with energy modeling and policy analysis. Questions include why the models were constructed, what questions they answer, historically the significant insights they have provided, how the models are being judged for credibility, and the extent to which these models are meaningful representations of the real world.

Forecasting national gas demand by modeling fuel purchasing decisions for end-use customer groups

by THOMAS R. RICE
Applied Decision Analysis
Palo Alto, California

and

C. JOHN MEESKE
Private Consultant
Troy, Michigan

INTRODUCTION

Rapidly rising domestic natural gas prices and the development of higher-priced synthetic fuels have made marketability a key issue for the natural gas industry, both for distribution companies and for producers and interstate pipelines in making reserve acquisition decisions. This paper describes the methodology underlying a detailed model that characterizes the interfuel competition between gas, oil, coal, purchased steam, and electricity.

The model forecasts the quantity of various fuels demanded in a utility company's service territory by considering the fuel-purchasing decisions of customer groups. Each market is segmented into customer groups according to equipment types and operating characteristics. For example, residential customers are divided according to their heating system (forced air systems, gravity systems, boilers, and heat pumps). Commercial customers are divided by Standard Industrials Classification (SIC) codes such as restaurants, laundries, schools, and hospitals. The industrial customers are also divided by SIC codes, and further subdivided by major installations. In some cases, both commercial and industrial customers are further broken down by actual device. For example, large boilers at a major industrial power plant form a separate market segment.

Consumers switch from one fuel type to another when it becomes economically desirable to do so, based on a net present value calculation that takes into account the following factors:

- "hard" economic costs associated with each fuel type
- subjective preferences for various fuels
- availability of alternate fuels
- efficiency of each fuel in providing usable energy
- relative importance of present and future costs to each consumer
- effect of government conservation programs and regulations
- risk of fuel shortages

Alternate gas rate designs included in the model provide for measurement of the impact of gas rates on gas demand. The model allows customers to switch to alternate (or multiple) fueled equipment when it is technically possible and economically desirable, taking into account the possibility that a consumer may be technically or contractually required to use a minimum amount of fuel with such equipment. The forecasted demand for each fuel type is produced by summing the demand for that fuel type for each consumer sector.

MODEL OVERVIEW

The major elements of the Interfuel Competition Model are shown in Figure 1. The central logic of the model deals with the manner in which customers expand the capacity of their energy consuming equipment and with their choice of fuel for both new and existing equipment. This central logic is supplied with information from several submodels. The Gas Rate Model computes the retail price of gas by sector based on projected cost of gas, sales volume, and distribution costs. Alternate fuel prices for coal, oil, and electricity depend on sulfur content, annual usage, and proximity to rail and water transportation. The Industry Growth Model considers economic growth by SIC code, the corresponding impact upon energy growth, and the relative growth of different technologies within a sector.

The Capacity Growth Model contains logic to deal with new energy consuming capacity added to keep pace with changes in economic activity level. It calculates the amount of energy consuming capacity existing, added, retired, and refurbished within each sector for each year under consid-

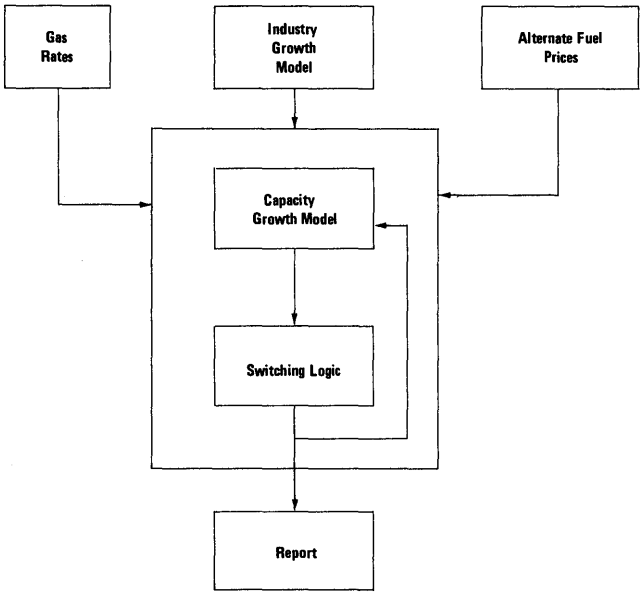


Figure 1—Overview of interfuel competition model

eration. Refurbishment is important since it represents a point where the incremental cost of converting fuel types is low.

The heart of the Interfuel Competition Model is the Switching Logic shown in Figure 1. This logic analyzes consumer decisions regarding possible fuels to meet energy needs during each of the 20 years considered in the study. The customer's decision is based on minimizing the net present cost associated with each possible fuel, including the cost of converting from one fuel to another as well as future prices of various fuels. Net present cost (NPC) is one component of the more general quantity, net present value (NPV). The net present value of a stream of investments, returns, and variable costs at some time period t is given by:

$$\text{NPV}_t=(1-\text{INCOME TAX RATE})\sum_{j=t}^{N}\frac{R_j-VC_j}{(1+)^{j-t}} - (1-\text{INVEST. TAX CREDIT})\sum_{j=t}^{N}\frac{I_j}{(1+r)^{j-t}} \quad (1)$$

where I_j is any investment made in time period j, R_j is the gross return or revenue in period j, VC_j is the variable cost in period j, r is the discount rate, and N is the decision maker's planning horizon.

This equation states that the net present value is equal to the net after-tax value in each period (revenue minus any investments, all weighted by the appropriate tax rates), discounted by the appropriate rate and summed over all time periods. The equation for the net present value can be written as the difference of two quantities, the net present return (NPR) and the net present cost (NPC):

$$\text{NPV}_t=\text{NPR}_t-\text{NPC}_t \quad (2)$$

The net present return is equal to the sum of the discounted returns in each time period weighted by 1 minus the income tax rate:

$$\text{NPR}_t=(1-\text{INCOME TAX RATE})\sum_{j=t}^{N}\frac{R_j}{(1=r)^j} \quad (3)$$

The net present cost is equal to the discounted variable costs and investments in each time period, weighted by the appropriate tax rates:

$$\text{NPC}_t=(1-\text{INCOME TAX RATE})\sum_{j=t}^{N}\frac{VC_j}{(1+r)^{j-t}} +(1-\text{INVEST. TAX CREDIT})\sum_{j=t}^{N}\frac{I_j}{(1+r)^{j-t}} \quad (4)$$

If a consumer switches from one fuel to another and continues to use the same amount of energy, the benefit or return that he derives from that energy in each period will be unchanged. Thus, the net present return does not depend on the fuel chosen.*

However, switching from one fuel to another will probably require a capital investment determined by the type of fuel selected; the variable costs associated with each fuel will be different.

The variable cost associated with using a particular fuel in time period j can be written as follows:

$$VC_j=[P_j(1+\text{CONVENIENCE PREMIUM}) +(\text{O\&M})_j]/\text{FUEL EFFICIENCY} \quad (5)$$

where P_j is the price of the fuel in time period j, $(\text{O\&M})_j$ is the operations and maintenance cost associated with using the fuel in time period j. "Convenience Premium" is a subjective term used to scale the actual price of a fuel up or down to reflect a consumer's bias for certain fuels. This subjective preference can be based on aesthetics, perceived safety, or relative ease of contracting. Since the net present return is the same for each unit of energy, regardless of the fuel chosen, a consumer can maximize his net present value by choosing the fuel that minimizes his net present cost. This is the economic criterion that is used in the Interfuel Competition Model.

Although the above equations give the correct definition for the net present cost associated with a particular type of fuel, a more convenient form is used in the Interfuel Competition Model. This definition is recursive; the net present cost in one time period is defined in terms of the net present cost in the following period:

$$\text{NPC}_t=(1+\text{INCOME TAX RATE})VC_t +(1+\text{INVEST. TAX CREDIT})I_t+\frac{\text{NPC}_{t+1}}{(1+\text{r})} \quad (6)$$

Although Equation (6) is equivalent to Equation (4) above, this equation for the net present cost does not require a sum of variable costs and investments over time. However, to

* This assumes that by switching from one fuel to another, a consumer does not face the risk of finding himself with an inadequate energy supply. The possibility of a general energy shortage and its impact on the decision making of energy consumers is discussed later.

calculate the net present cost in any time period with this equation, we must first calculate the net present cost in the following time period. In the Interfuel Competition Model the net present costs associated with each fuel and each time period are computed by working backwards through time, starting with the last year considered by the model and working backwards to the present.

DECISION PATH OUTPUT FROM SWITCHING LOGIC

The Interfuel Competition Model calculates the net present cost associated with switching from every possible fuel to every other possible fuel (or keeping the same fuel) in each year. The program then selects the fuel that minimizes the customer's net present cost. Using this information, the model produces an array of decisions showing which fuels the customer should use over time for each possible initial fuel type. This array of decision for a hypothetical customer is represented graphically in Figure 2.

If the consumer was initially using gas, the decision arrows in Figure 2 indicate that he should continue to use gas through the third year and then switch to oil. On the other hand, if he started with coal, he should continue to use his initial fuel for at least the first four years. Figure 2 also indicates that customer using gas in year 4 should switch to oil, even though a customer who had started with gas in the initial year would have switched to oil the third year. Thus, if the decisions shown in Figure 2 are followed, it would not be possible for this customer to be using gas in the fourth year.

ILLUSTRATIVE EXAMPLES OF THE SWITCHING LOGIC

The model's switching logic can best be illustrated by simple examples. To keep the examples manageable, fuel choices are limited to gas (G), oil (O), coal (C), and dual-fired gas and oil (G/O). In the actual model additional fuel types are possible, including electricity, purchased steam, dual-fired gas and coal. In addition, we will ignore the distinction between refurbished and remaining equipment in these examples.

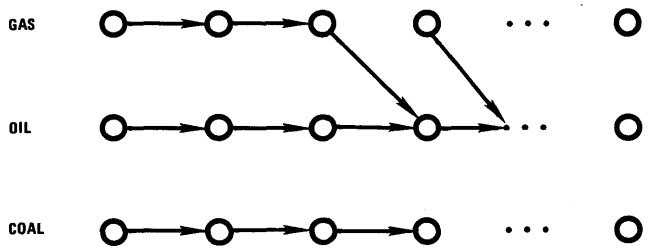


Figure 2—Decision paths—Switching logic output

Example without fuel availability uncertainty

The decisions that a customer with four possible fuel types must face in any given year are shown in Figure 3. At the end of year the customer could be using any one of the four possible fuels, as indicated by the four hexagons at the left of Figure 3. In fact, he may have several different types of equipment using different fuels. Some of his existing equipment will have to be retired, and he may need to add some new equipment to his inventory. For each type of equipment—new, currently using gas, currently using oil, currently using coal, and currently dual-fired gas/oil—he must decide which fuel to use in the future. Decisions are shown in Figure 3 by small squares. Separate decisions are made for new equipment and for equipment currently using each of the possible fuels because the economics of switching fuels is different in each case. Once the customer has made all decisions for year I, the various pieces of energy consuming equipment can be aggregated by fuel type. Thus, if the customer decides to continue using gas for his gas-fired equipment, and to purchase some new equipment that uses gas, the amount of gas-fired equipment carried over to the next year will include the total of both these equipment categories. In the following year, I+1, the customer, after retiring some of his old equipment and purchasing some new equipment, is again faced with the same set of decisions. In each case the decisions are based on minimizing the net present cost associated with each possible fuel.

Figure 4 illustrates how the consumer might make his decisions for new equipment, equipment currently using gas, and equipment that can use either gas or oil. It is assumed that the consumer starts in year I−1 with ten units of capacity in gas-fired equipment that can burn either gas or oil, as represented by the 10's above the hexagons at the left of Figure 4. If four new units of capacity are purchased to replace two units of capacity for each type of equipment which must be retired, the three decisions for year I, represented by the boxes at the left are: What is the best fuel type for: (1) the new equipment, (2) the eight units of capacity currently using gas, and (3) the eight units of capacity currently using gas and oil?

The Interfuel Competition Model treats equipment capable of using more than one fuel (such as the gas/oil equipment in this example) as if it were a separate fuel type. After the model has allocated demand to dual-fuel equipment, a separate calculation determines the cheapest available fuel that the equipment can use in each year. Thus, gas and gas/oil equipment may both use gas in a particular year. In this case the allocations to gas and gas/oil equipment must be summed to determine the total gas demand. However, if oil becomes less expensive than gas in the following year, the dual-fueled gas/oil equipment will automatically switch to oil without incurring any additional investment costs. Equipment based on gas alone will switch to oil only if the discounted cost savings are sufficient to outweigh the required investment.*

* Multiple-fuel equipment may be required to use a minimum amount of each of its possible fuels.

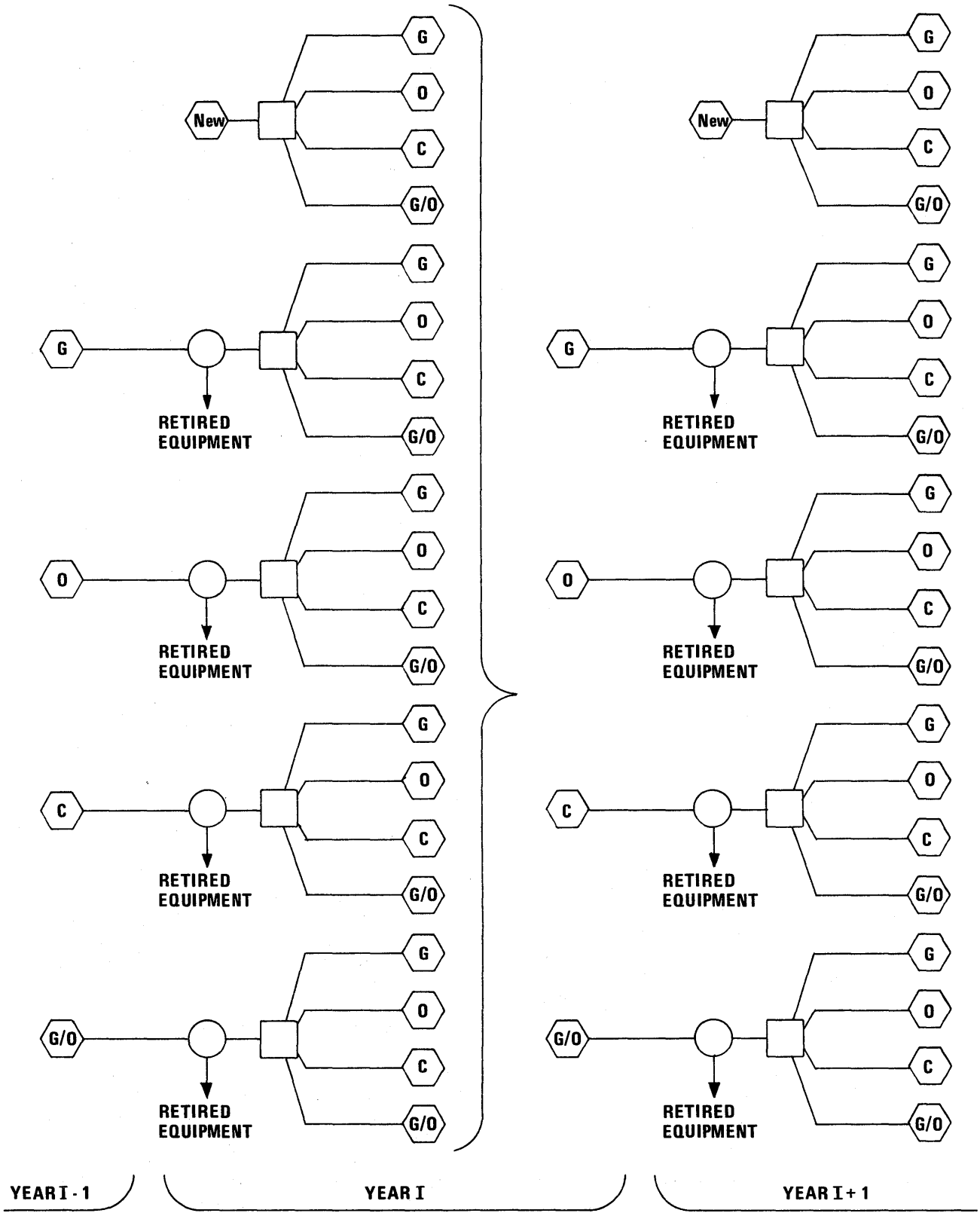


Figure 3—Basic switching logic

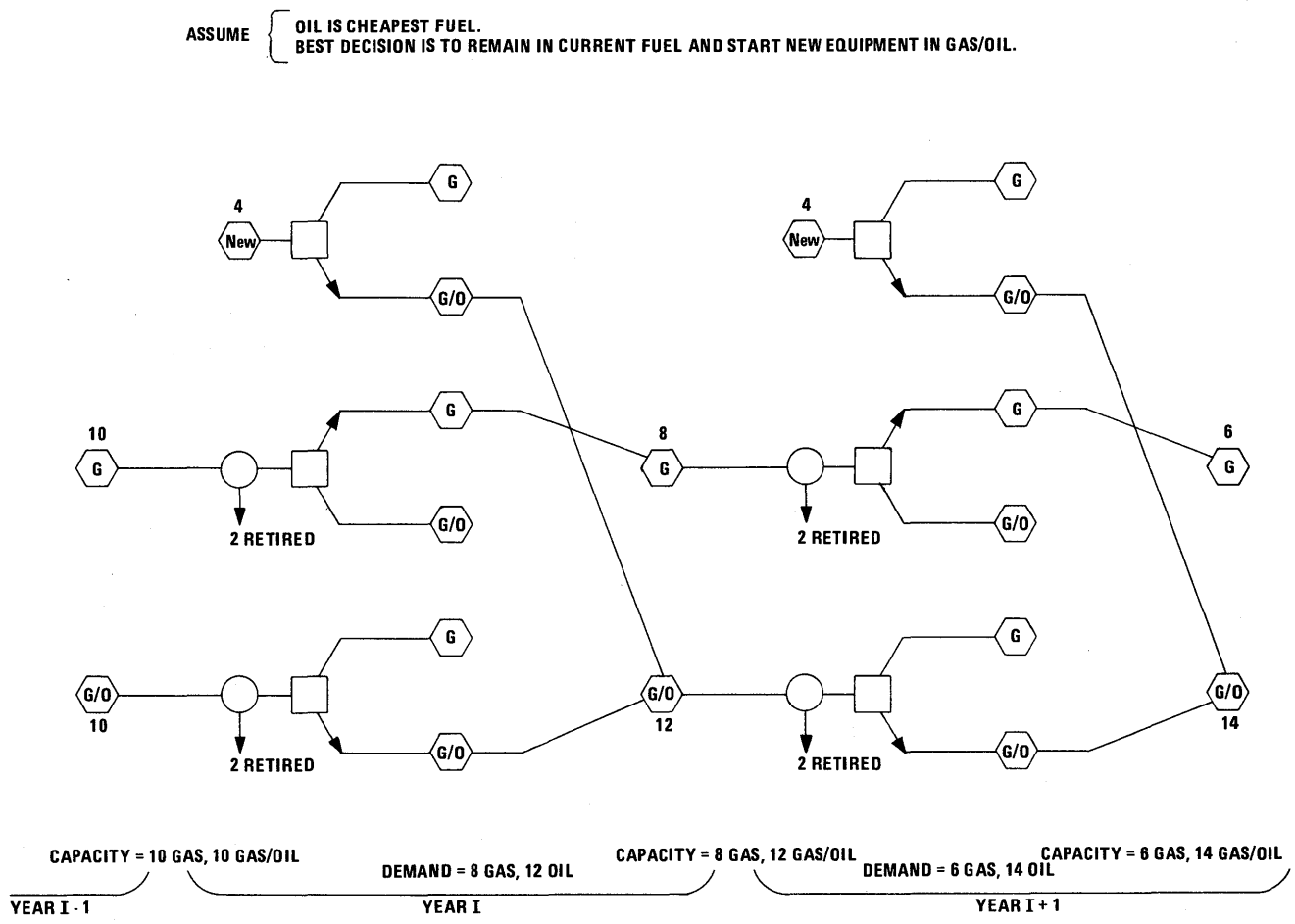


Figure 4—An example of the basic switching logic

Let us assume that oil is the cheapest fuel and that the best decision at each point is to continue using the existing type of equipment or to purchase new equipment that is capable of burning both gas and oil. The consumer's decisions and their effect on the capacity of his equipment and his demand for various fuels is shown in Figure 4. In year I−1 two of the initial ten units of capacity in equipment using gas only are retired, and the remaining eight units continue to burn gas only. Similarly, eight of the initial ten units of capacity in dual-fuel gas/oil equipment are maintained through year I and combined with four new units of capacity capable of burning both fuels. Thus at the end of year I there are eight units of capacity in gas only and twelve units of capacity in gas/oil. Since the dual-fuel equipment is used with the cheapest fuel, oil, the fuel demand in year I is for eight units of gas and twelve units of oil. In year I+1 the process repeats itself, with the capacity of dual-fired equipment increasing from twelve to fourteen.

Example with uncertain fuel availability

To generalize the example to uncertain fuel availability we must differentiate between three general categories of unavailability. The first is *operational reliability*; this can be modeled by adjustments to O&M expenses to reflect inventory carrying costs and stockouts, as in any inventory problem. The second is *long-term shortages* of specific fuel types caused by government regulations or scarcity; this uncertainty is treated explicitly in the model by introducing the probability of unavailability as illustrated in the example below. The third is a *general energy shortage*, where total supply is insufficient to meet demand. This is modeled by adding a term to the variable cost equal to the product of three terms: (1) the probability that a general energy shortage will occur in each future year, (2) the probability that a consumer will be forced to do without energy if he is using a particular fuel when an energy shortage occurs, and (3) the net cost to the consumer that results from doing without energy for the expected duration of the shortage.

Figure 5 illustrates a simplified numerical example of the switching logic contained in the Interfuel Competition Model when fuel availability is uncertain. Only two types of equipment are considered in the example: that using gas only, and that capable of burning either gas or oil. For simplicity, it is assumed that there is no new, retired, or refurbished equipment; thus all of the existing equipment is carried over from one year to the next without additions, losses, or sep-

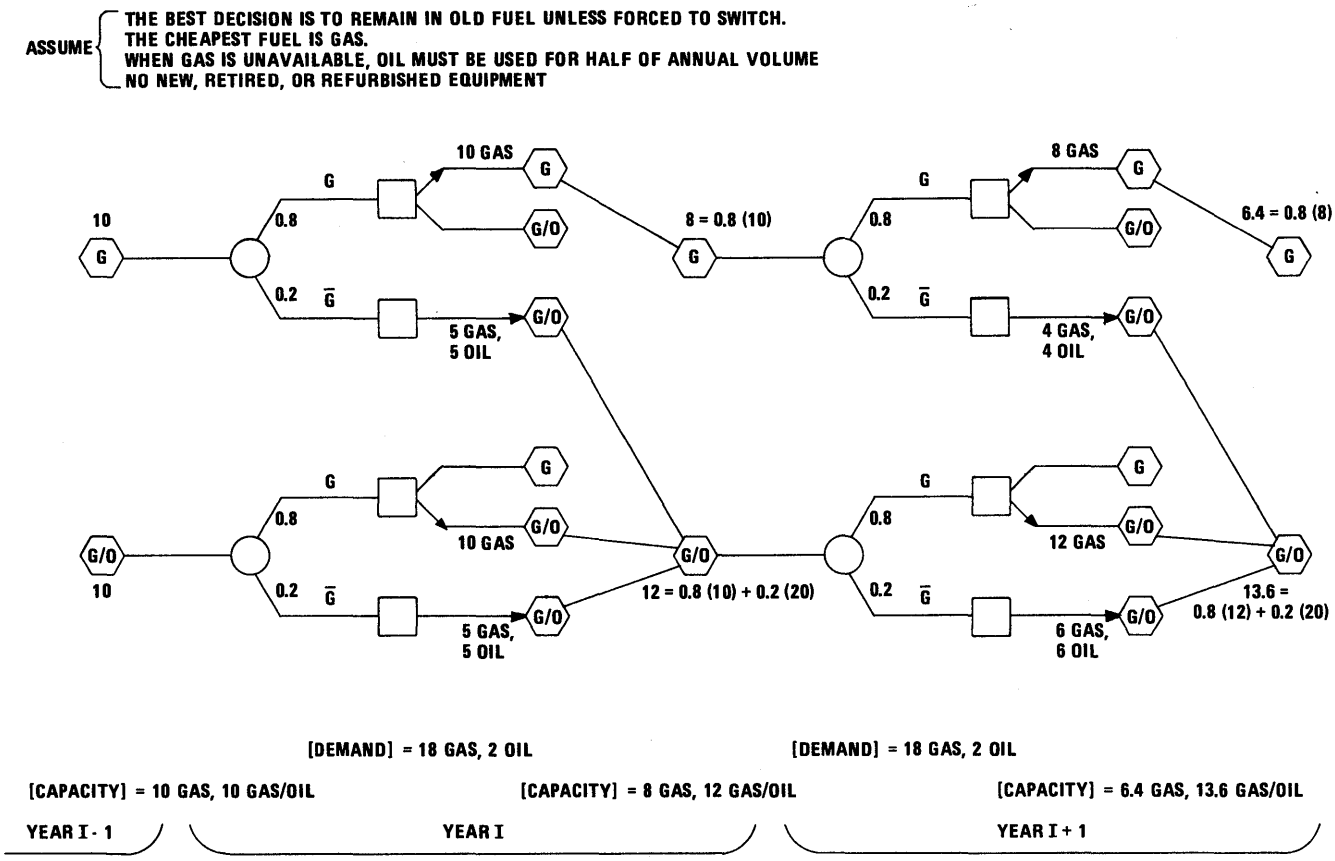


Figure 5—An example of switching logic with uncertain fuel availability

arate decisions for refurbished equipment. It is also assumed that: (1) the best decision in each year is to remain with the current fuel unless that fuel becomes unavailable, (2) gas is the cheapest fuel, and (3) in a year when gas is unavailable, the duration of the shortage will force consumers to use oil for half their annual volume. In addition, it is assumed that there is a 20 percent chance that gas will be unavailable in each year. This is illustrated in Figure 5 by the probability node shown as a circle.

The example starts at the end of year (I−1) with ten units of equipment capable of using gas only and ten units of equipment capable of using either gas or oil. In this case, the ten units of dual-fired equipment are run on gas since it is the cheapest fuel. If gas is available throughout the year I, none of the equipment is modified and twenty units of gas are demanded. However, if gas is unavailable in year I, the ten units of equipment on gas only are converted to allow them to burn either gas or oil. All the dual-fueled equipment is then operated on oil for half the annual volume, and on gas for the rest of the year. At the end of year I, either ten units of capacity will operate on gas and ten on gas/oil, or all of the equipment will have been converted to gas/oil, depending on whether or not gas was available during the year. The expected capacity at the end of year I is eight units of equipment capable of burning gas only, and twelve units of dual-fuel equipment. During year I there is an 80 percent that the entire demand will be for gas, and a 20 percent chance that the demand will be split evenly between gas and oil. Thus, the expected demand will be split evenly between gas and oil. The expected demand during the year is for 18 units of gas and two units of oil.

In the year I+1, the process repeats itself. The expected capacity of equipment capable of burning only gas drops from 8 to 6.4, while the expected capacity of dual-fuel equipment rises from 12 to 13.6. However, there is still an 80 percent chance that the entire demand during year I+1 is for 18 units of gas and two units of oil, unchanged from the previous year. This example illustrates the fact that the possibility of fuel shortages may cause consumers to protect themselves by switching to dual-fired equipment without changing the expected demand for individual fuels. This type of consumer behavior, which is incorporated into the Inter-fuel Competition Model, leads to volatile energy markets marked by large fractions of total energy demand that switch rapidly from one fuel to another.

OTHER ISSUES

Fuel availability is just one of many issues considered in designing and implementing the Interfuel Competition Model. Other phenomena of importance include:

- Lags in consumer response to energy market changes
- Different planning horizons used by energy consumers
- Legal and physical constraints that limit a consumer's ability to switch from one fuel to another
- Rate structures employed by electric utilities
- Effects of conservation, regulation, and growth within the gross national product on energy demand
- Interdependence of prices and volumes of all fuels.

CONCLUSION

This model is among the first models to predict market demand and market share by examining the behavior of specific customer groups. Most other demand forecasting methodologies rely on price elasticities or market share curves, and, consequently, they predict a smooth response of demand to price differentials between fuel types. Of course, price response depends on specific data used in the model, but typical demand curves derived from the model are not smooth. Gas demand forecasts from the model based on dynamic programming, probabilistic decision theory, and detailed market segmentation should be more accurate than forecasts from traditional models.

General equilibrium models for energy policy analysis

by HORACE W. BROCK
SRI International
Menlo Park, California

INTRODUCTION

In this paper we shall provide an elementary overview of the concept of general economic equilibrium in the concrete context of the U.S. energy market.* With this aim in mind, we shall introduce and discuss several important, large-scale energy models which embody the assumptions of the methodology of economic equilibrium. The models which will be discussed were all developed since the OPEC embargo of 1973—an event which has provided the impetus for development of sophisticated computer models useful in energy planning and policy-making.

In the first section, the models of Hudson and Jorgenson[1] and Hnyilicza[2] are introduced. The discussion here will hopefully provide the reader with a clear diagrammatic exposition of the concept of general economic equilibrium. By "economic equilibrium" we mean a state of the economy in which prices are such as to equate the supply and demand for all commodities under consideration. These two particular models provide a good starting point for our discussion since they are very broad in scope, encompassing both the energy and the non-energy sectors of the U.S. economy. In doing so they take account of the important linkages which exist between the energy and non-energy sectors.

Then in the second section, we introduce the SRI National energy model, sometimes referred to as the SRI-Gulf model since it was originally developed with the collaboration of the Gulf Oil Corporation. Our focus in this section is on the *algorithm* through which economic equilibrium is determined in a dynamic economic context. An understanding of the algorithm not only sponsors an in-depth understanding of the SRI model and "how it works," but permits an appreciation of the important role of *computation* in large-scale models. Finally, in the last section, we conclude with some remarks on the role of economic models in energy policy analysis.

* The research discussed in this paper was sponsored by the Office of Policy Research and Analysis of the U.S. National Science Foundation. The report[3] on which this paper is based was co-authored by the present author, and by Dr. Dale M. Nesbitt of Decision Focus, Inc. of Palo Alto, California.

THE HUDSON-JORGENSON/HNYILICZA MODELS—AGGREGATE MODELS OF ENERGY AND ECONOMIC GROWTH

Background

In 1974, Edward Hudson and Dale Jorgenson[1] introduced a model of energy and economic growth. The Hudson-Jorgenson model consisted of two somewhat separate submodels: a macro-economic model of economic growth, and a multi-sector interindustry model. The model was the first to make use of a new and advanced methodology for econometric estimation in a general economic equilibrium context. This methodology is known as "translog economic modeling" in the technical literature.

An ostensible problem with the Hudson-Jorgenson model was its failure to integrate its two submodels in a completely satisfactory way. It was not clear that the inputs and outputs of the two submodels were compatible. This difficulty provided the motivation for Esteban Hnyilicza[2] to develop a model which, while very similar to the Hudson-Jorgenson model, differed from it in its treatment of economic growth. Specifically, Hnyilicza has attempted to build a model in which the multi-sector inter-industry model and the macro-economic growth model are satisfactorily integrated. The Hnyilicza model also differs from the Hudson-Jorgenson model in being much more highly aggregated. Specifically, it consists of only two sectors, namely the energy and non-energy sectors of the economy. In other important respects, such as use of the "translog" methodology, the two models are much the same.

In this section, we are going to discuss the Hnyilicza model at some length. The purpose of discussing this model rather than the Hudson-Jorgenson model is that our diagrammatic mode of exposition of the methodology used in both models requires that we limit ourselves to a simple model with very few sectors.

Introduction

Hnyilicza has constructed an integrated model for the purpose of analyzing the relationship between economic

growth, and the (equilibrium) prices and quantities of both energy and non-energy goods. His model provides the basis for computing a trajectory of future market clearing (i.e., "equilibrium") prices and quantities of the following commodities:

1. Energy consumption goods
2. Non-energy consumption goods
3. Energy intermediate goods
4. Non-energy intermediate goods
5. Capital services to the energy sector
6. Capital services to the non-energy sector
7. Labor services
8. Energy imports
9. Non-energy imports

These outputs are generated on a period-by-period basis as the solution to a model consisting of fifty-five (nonlinear) simultaneous equations. By analyzing time-series projections of the prices and quantities of these goods, Hnyilicza can diagnose the relationship between economic growth on the one hand, and the relative prices of energy and non-energy goods on the other hand. It is also possible to study the rate of substitution of capital and labor for energy, as well as the economic impact of alternative tax and conservation policies.

Overview of the model

Figure 1 displays the overall structure of the Hnyilicza model. There are three basic model sectors: industrial production, consumption, and investment.* The fact that these sectors are enclosed by solid as opposed to dashed lines is supposed to describe the fact that the outputs of these sectors are determined endogenously within the model. This is not the case with the government sector and the foreign sector. The latter two (which are enclosed by dashed lines) are largely exogenous to the model.†

Before delving into the details of the three sectors, let us briefly summarize what takes place in the model in hopes of better motivating Figure 1 and in hopes of providing the reader with a guide to the model. One preliminary word of warning is in order. It is extremely difficult to describe what "goes on" in any general equilibrium economic model on a sector-by-sector basis. The reason for this is simply that what happens in any one sector affects every other sector. Nonetheless, we have tried to motivate the model by means of a sector-by-sector discussion since there seems to be no satisfactory expository alternative.

Two things happen in the consumption sector. First, the "national household's" inter-temporal utility function determines the split between *present* consumption and *future* consumption (i.e., savings). More specifically, the household is assumed to determine a utility-maximizing split between consumption and savings as an (econometrically estimated) function of wealth, the wage rate, and government transfer payment levels. Second, the consumer decides upon an *intra*-period split between leisure, energy consumption goods, non-energy consumption goods, and capital services. The optimal split is that which is utility-maximizing, subject to the consumer's budget constraint. This split will of course be a function of the relative prices of the various consumption goods, and the wage rate, among other things.

The two industrial production sectors—energy and non-energy—determine an optimal (profit-maximizing) level of outputs, *and* an optimal (profit-maximizing) configuration of inputs used in producing the equilibrium output levels.

Finally, the investment sector determines the level of capital services which will be available in any given period. As we shall see, this level depends on both the efficiency of capital, and on the level of capital stocks in the preceding period. The investment sector also determines the overall level of gross investment in any given period, as well as the fractions of gross investment going to the energy sector, the non-energy sector and the household.

A diagrammatic exposition

An important question arises concerning the best way in which to explain and motivate a model which is as general as the Hnyilicza model. Two approaches seem to be possible: a mathematical approach and a diagrammatic approach. We have settled upon a diagrammatic exposition of the model. More specifically, we shall motivate the model by discussing each of the three sectors (production, consumption, and investment) from the standpoint of the *demand curves* and the *supply curves* which are implicit in the equations which characterize economic behavior in each sector. This strikes us as a compact and intelligible mode of exposition. Furthermore it permits us to conclude our discussion with what we hope to be a highly appealing summary of the model: namely, a picture (Figure 5) in which we superimpose the various supply and demand curves which we "extract" from each of the three sectors. The price-quantity pairs of *all* of the points of *intersection* of these supply and demand curves will clearly constitute a general economic equilibrium. And these equilibrium prices and quantities coincide with the values of the endogenous variables which are determined when the model is solved.

In light of our decision to explicate the model in terms of the supply and demand relationships that are implicit in the equilibrium equations of the model, a caveat must be issued from the outset. Whenever the reader sees a picture of a supply curve or a demand curve, he should realize that he is beholding an illusion. For in point of fact, there are no such things as supply and demand curves *per se*. There are only *conditional* supply and demand curves. Let us illustrate this point since it often engenders confusion. Consider a two

* It is perhaps a bit artificial and unnecessary to identify an "investment sector" which is distinct from the consumption and production sectors. We have only done so in order to facilitate a description of what goes on in a "growth" model.

† Specifically, the behavior of the government sector is almost completely exogenous. In the import markets, whereas the supply curve for imported goods is exogenous, the demand curve for imports (energy and non-energy) is endogenous.

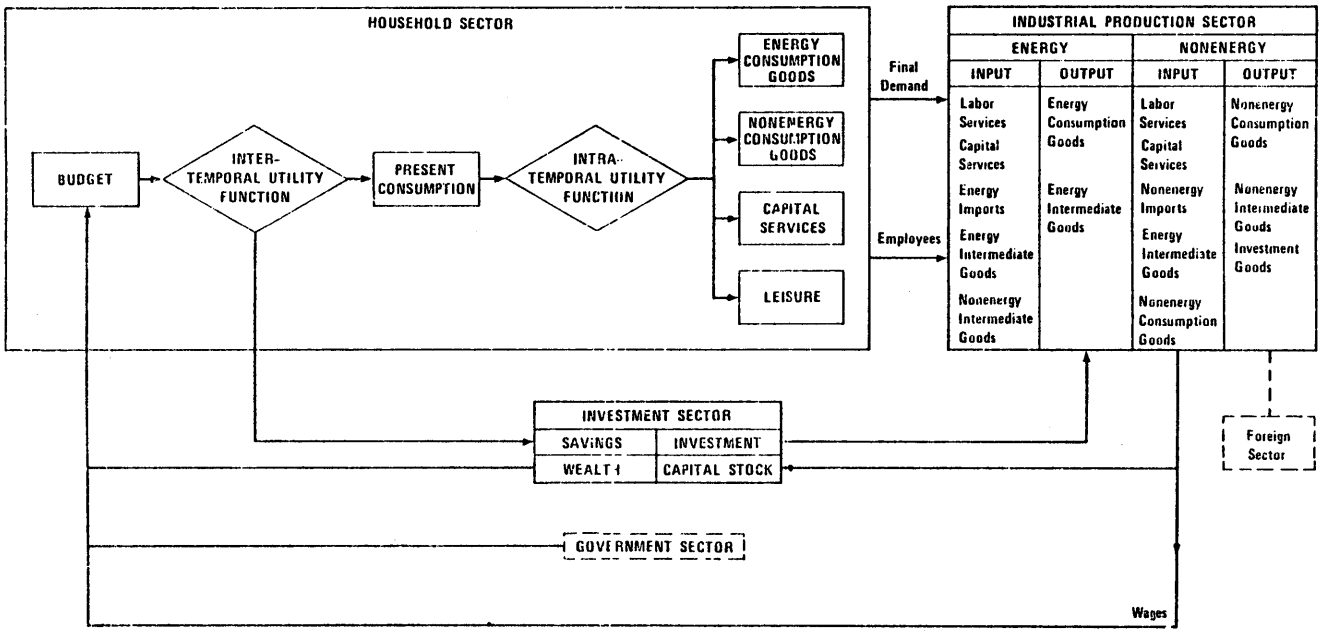


Figure 1—The Hnyilicza model structure

commodity world consisting of coffee and sugar. In general, the reader would be hard pressed to spell out his price-demand relationship for coffee. He would only be able to do so if he knew how much sugar he would be allotted. Thus we can speak of a demand curve for coffee *conditional* upon a particular consumption level of sugar. For this reason, it is somewhat artificial to characterize market equilibrium below in terms of a set of supply curves and demand curves.‡

Methodology of the model

Industrial production sector

The function of the production sector in the context of the larger model is twofold:

a. to determine an equilibrium factor input mix which is optimal (profit maximizing) for each industry in producing its total output;
b. to determine the total (equilibrium) output levels that meet the demands of the intermediate market *and* of the final goods market.

As we saw in Figure 1 and as we see in more detail in Figure 2 on the next page, we can think of the production sector as an input-output structure. We summarize this input-output structure in Table I.

‡ Nonetheless, it will be true that *if* we somehow knew the equilibrium quantities of all commodities, then we could sketch a set of conditional supply and demand curves whose points of intersection *do* constitute the market equilibrium. The problem of course is that one does not know the equilibrium in advance.

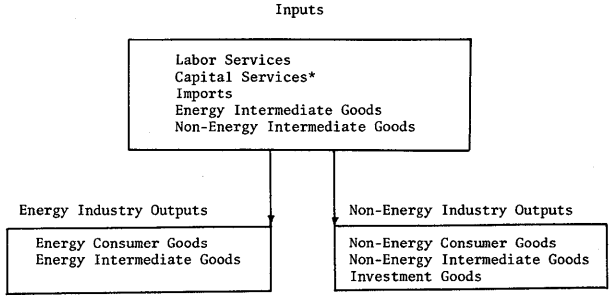
It is important to realize that even though we are speaking of production in "input-output" terms, the Hnyilicza model is not an input-output model in the classical sense of that term. Rather, it is a *generalized input-output* model. In such models the input-output (transactions) coefficients, rather than being given exogenously (as would be the case in the economy whose technologies permitted no substitutability among inputs), are *functions* of the prices of all inputs *and* the configuration of final demand. Specifically, we can write

$$a_{ij} = f(w,p,y,x)$$

where: w is the vector of input factor prices
p is the vector of output prices
y is the vector of final demands
x is the vector of gross outputs.

The *equilibrium* coefficients $\{a_{ij}^*\}$ that are implicitly solved for by the model designate the profit-maximizing

TABLE I



*Capital services are the annual services generated by existing stocks of capital goods. These services are provided to the productive process. "Capital goods" have a life exceeding one year, by definition, whereas "intermediate goods" are consumed by the production process within a year.

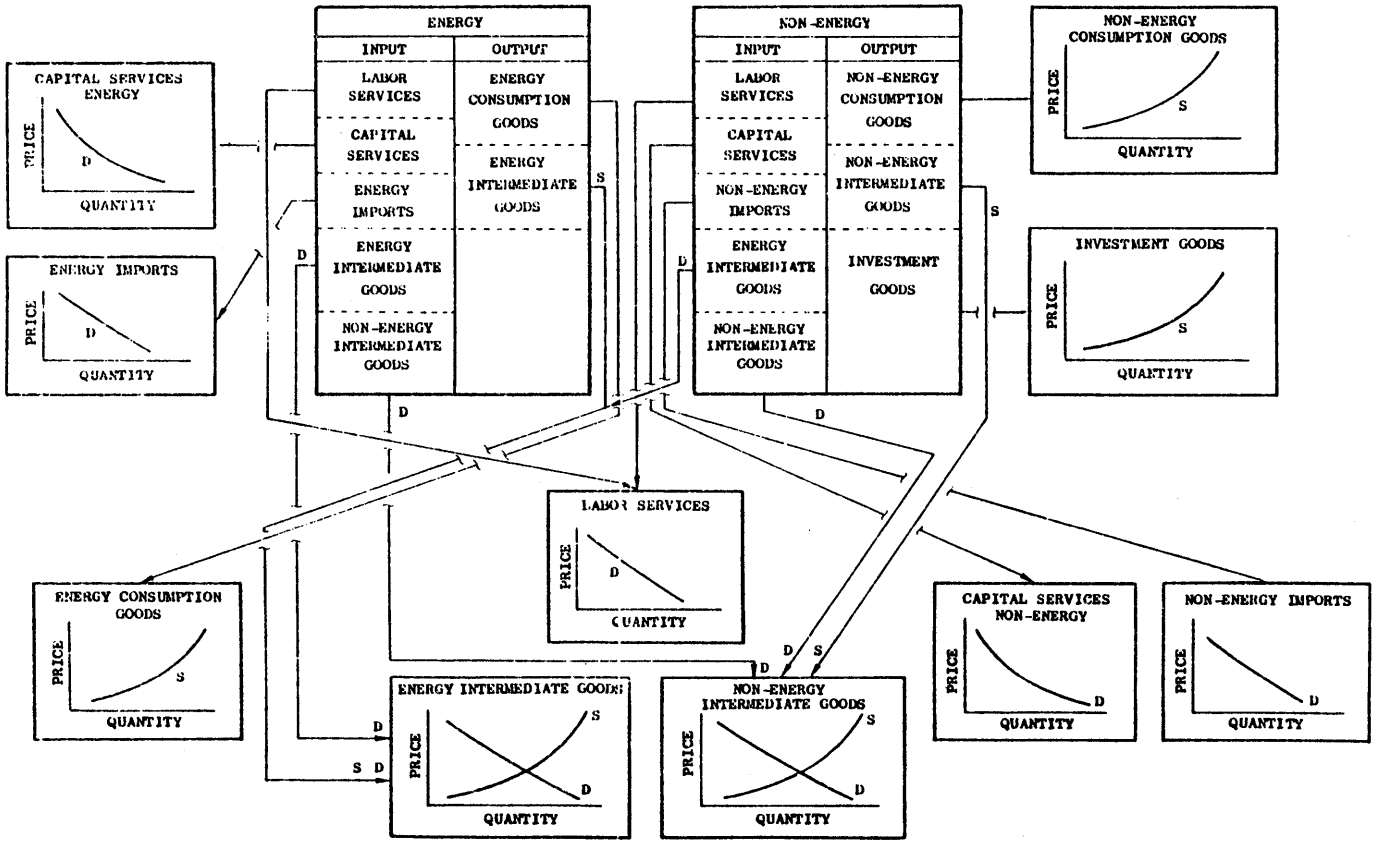


Figure 2—Industrial production sector

number of units of commodity i which are utilized in the production of each unit of commodity j. Let us now press towards a deeper understanding of what is going on in the production sector. A result of this will be an enhanced understanding of the equilibrium coefficients $\{a_{ij}^*\}$.

The most straightforward way in which to characterize what goes on in the production sector of the model is to depict the various supply curves and demand curves which are implicit in the equilibrium equations which characterize rational economic behavior on the part of the two industries: energy and non-energy.

The entrepreneur must make essentially two different types of decisions. He must decide upon a profit maximizing *level of output* of the commodities he manufactures; and he must decide upon a profit maximizing *mix of inputs* to be used in meeting his (optimal) production schedule.

Clearly his optimal choice of output levels and input mixes will be a function of the relative prices of outputs and inputs. Indeed, it is customary to speak of the relationship between the price of a given input and the quantity demanded by the entrepreneur at that price as the *derived factor demand curve* of the input (or "factor") in question. And the relationship between the optimal output level of a given output and the price of the output is summarized by the *supply curve* of the output.

With this in mind the reader is referred to Figure 2, where we identify the various derived demand curves and supply curves that are generated within the production sector. Let us summarize the information contained in the figure.

The derived factor demand curves and the supply curves

There is one derived factor demand curve for each input. Since both the energy and the non-energy sector use five inputs, there are therefore ten derived demand curves in all:

	Energy	Non-Energy
Demands:	Labor Services	Labor Services
	Capital Services	Capital Services
	Energy Intermediate Goods	Energy Intermediate Goods
	Non-Energy Intermediate Goods	Non-Energy Intermediate Goods
	Imports	Imports

The derived demand curves associated with these input demands are illustrated in Figure 2.

With respect to supply, the energy sector produces only two goods whereas the non-energy sector produces three

goods:

	Energy	Non-Energy
Supply:	Energy Consumption Goods	Non-Energy Consumption Goods
	Energy Intermediate Goods	Non-Energy Intermediate Goods
		Investment Goods

The associated supply curves are indicated in Figure 2.

An input-output interpretation

We are now in a position to give an interpretation to the equilibrium Input-Output coefficients $\{a_{ij}^*\}$. Note in Figure 2 that in two (and only in two) of the markets pictured there do we have well-defined supply and demand curves. This is the case in the markets for energy intermediate goods and non-energy intermediate goods. Suppose now that we could "complete" our description of the remaining eight markets by obtaining the missing supply and demand curves. (As will be seen, these missing curves are generated in the other two sectors of the model to which we shall turn shortly.) Suppose finally that we were to take note of the equilibrium point in these ten markets, that is, the equilibrium prices and quantities of the ten commodities.

It would then be straightforward to compute the equilibrium input-output coefficients either in dollar terms or in physical terms—using simple arithmetic. The collection of coefficients so obtained would coincide with the collection $\{a_{ij}^*\}$. That is, these are the equilibrium input-output coefficients associated with profit-maximizing behavior on the part of the industries with respect to the choice of input mix and the choice of output level.

The consumption sector

Figure 3 portrays the consumption sector in some level of detail. Looking at the top part of the figure it is clear that two things take place in the consumption sector model. First, an "intertemporal" utility function determines a utility-maximizing split between present and future consumption. This utility function is an econometrically estimated function of:

Time
Interest rate
Existing wealth
Wage rate
Government payments to household (e.g., welfare)

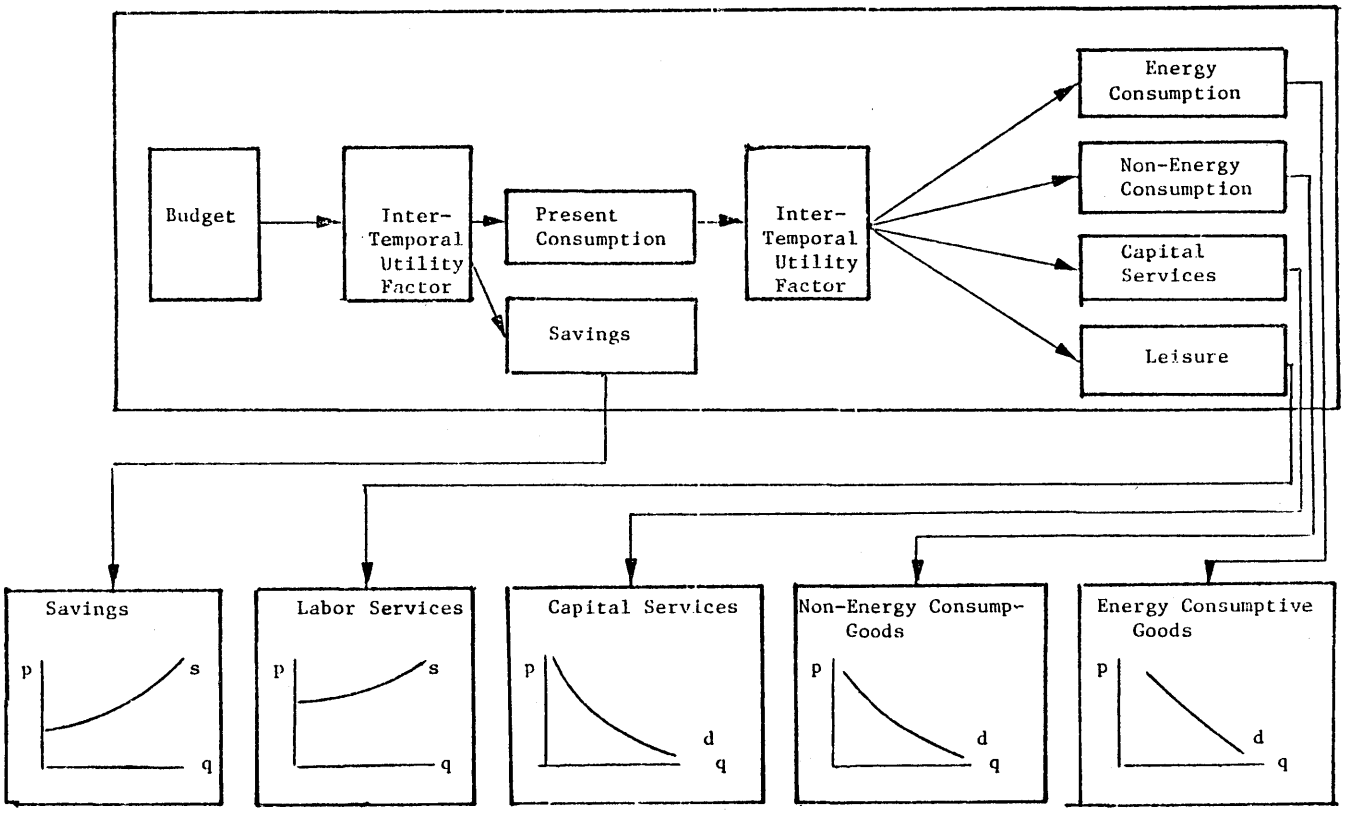


Figure 3—The household sector

Second, an "intratemporal" utility function determines the split *within* a given period of household expenditures between the four consumption goods, namely energy consumption goods, non-energy consumption goods, capital services, and leisure. By capital services we mean the annualized value of the services which flow from consumer durables or residential stock. The intratemporal utility function which performs this split is an econometrically estimated function of price, the level of consumption of each good, and time.

A supply and demand curve interpretation

A deeper understanding of the consumption sector can be obtained by considering the supply and demand curves which are implicit in the equilibrium equations characterizing utility-maximizing behavior on the part of the household. These price-quantity relations are shown in the lower portion of Figure 3. Interpretation of all of these curves is straightforward with the exception of the supply curve of labor services. This curve is in fact the inverse of the demand curve for leisure services. As the wage rate increases, leisure becomes more expensive. Work is substituted for leisure, which increases the supply of labor.

Investment and capital accumulation

Formally speaking, the investment sector is part of the production sector. Nonetheless, it is useful to isolate it as a separate sector for the purposes of enhancing an understanding of capital accumulation in the Hnyilicza model. The situation is pictured in Figure 4. Consider the upper part of the figure. The supply of new investment goods that come on stream in a given year t appears on the left. Next, the supply of new investment goods is split three ways into the shares of investment goods going to the energy, non-energy, and domestic sectors. This split is realized by the use of a multiplier which reflects historical ratios in the apportionment of investment goods.

Moving further to the right, we add in the depreciated value of capital stock for the respective sectors. These are the values of stocks at the end of the *previous* year. The result of this addition is the *current level of capital stock.*

Moving yet further to the right in Figure 4, the values of the current level of capital stock in the two industrial sectors are multiplied by the efficiency of capital. The result of this operation is defined to be the *level of capital services* in period t.

We exhibit the current supply of capital services generated in this fashion within each of the two industries as two supply curves. The reader will note that these supply curves are perfectly inelastic. The reason for this is that the level of capital services is not a function of any current prices. One might think that this level would be a function of the current price of new investment goods. This is not the case, for in this model there is a one period *lag* between the time when capital goods are produced and the time when they begin to generate capital services. The investment goods which come "on stream" in year k and which appear in the upper left of Figure 4 were actually produced in year $(k-1)$.

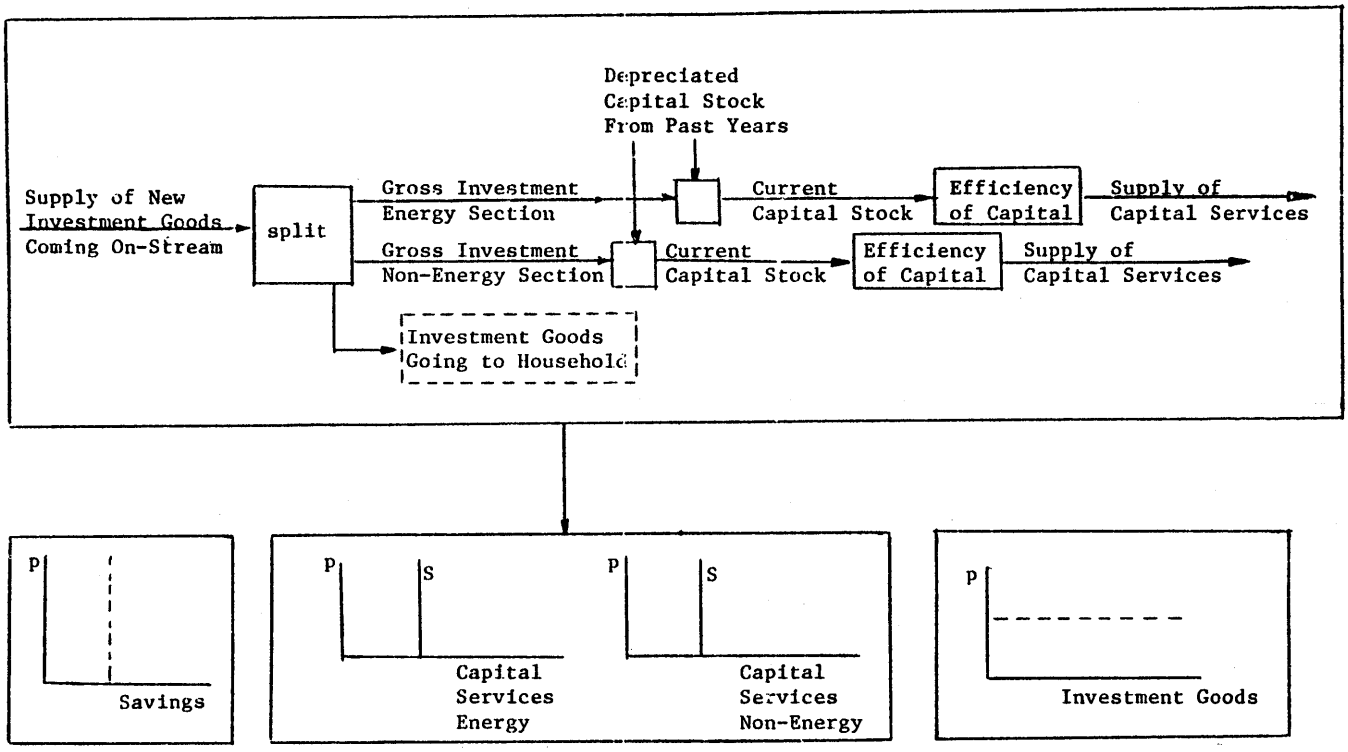


Figure 4—Investment sector

Two other price-quantity curves appear in the figure. First, there is a demand curve for investments goods. This demand curve is seen to be perfectly elastic. Moreover, it appears as a dashed line. Why? We have used a dashed line to indicate that this demand curve is an artificial construct. For in the Hnyilicza[2] model, the (equilibrium) level of investment goods is, in fact, "supply-determined." Whatever quantity of investments goods is supplied is *assumed* to be absorbed—irrespective of own price. It is for this latter reason that the curve appears as perfectly elastic.

In point of fact, it is possible to introduce a demand *function* for investment goods in the Hnyilicza model. For one of the basic equilibrium conditions of the model is that investment equals income minus consumption. [This equilibrium condition is imposed on the model.] Consumption, in turn, is a function of income and of the price of labor, capital, etc. Thus, the demand for investment goods is indeed a function of "prices" in the Hnyilicza model. Our point above is simply that it is not helpful to characterize a market of investment goods *per se,* and to plot the demand for investment goods as a function of the price for investment goods. For, as we said above, whatever quantity of investment goods is produced is assumed to be demanded.

Last of all, there is a demand curve for savings. This appears as a dashed vertical line. Why? The reason for this is that there is no "demand curve" for savings *per se* in the model. Rather, savings is *assumed* to equal investment. Once again, just as in the case of the demand for investment goods above, it is possible to assert that there is a demand *function* for savings embedded in the Hnyilicza model. For the so-called "static equilibrium condition" of this type of growth model is that savings equals income minus "desired" consumption. This relationship is imposed on the model, and serves to relate industry demand for savings to consumption and hence to relative prices, since consumption depends in part on prices.

The reader who finds the rather arbitrary treatment of investment and savings perplexing should be aware of the motivation for the static equilibrium conditions such as "Savings=Investment" that are imposed on the model. The rationale for these assumptions is simply to insure consistency between the aggregate savings decisions by individuals on the one hand, and the aggregate investment decisions of producers on the other hand. There would be no *need,* in principle, for imposing static equilibrium conditions if the model included a full set of futures markets, and if an equilibrium set of prices and quantities across time were computed simultaneously, as is the case in the SRI model. Of course, to point this out is not to make a value judgment about the quality of the SRI model versus the Hnyilicza model, since the emphases of the two models are different.

General equilibrium in the Hnyilicza model

Figure 5 shows the result of collecting together the various supply and demand curves that we have extracted from the production sector, the consumption sector, and the investment sector. The point of intersection of all of these curves coincides with the equilibrium prices and quantities that Hnyilicza's fifty-five simultaneous equations are solved for. We can thus interpret equilibrium in terms of supply-demand balance in the following markets:

1. Capital Services to the Energy Industry
2. Capital Services to the Non-energy Industry
3. Labor Services
4. Energy Imports
5. Non-energy Imports
6. Energy Consumption Goods
7. Non-energy Consumption Goods
8. Energy Intermediate Goods
9. Non-energy Intermediate Goods
10. [Investment Goods]
11. [Savings]

Time-series projections of these equilibrium prices and quantities are the principal outputs of the Hnyilicza model. The following variables are exogenous to the model and are required as inputs:

- Inventory investment
- Labor force
- Tax rates
- Government expenditures
- Exports
- International financial claims
- Government transfer payments (e.g., welfare)
- Rate of unemployment
- Import prices
- Rate of replacement of capital stock
- Initial year private national wealth
- Initial year capital stock
- Quality of capital (a constant, multiplied by capital stock to give capital services)

A note on the new version of the Hudson-Jorgenson model*

Hudson and Jorgenson have recently reworked their original model. Their new model is very similar to the Hnyilicza model, especially as regards the use of the variable coefficient input-output methodology in an economic equilibrium context. Where the two models differ is in the degree of disaggregation of the production sector. As we saw, the Hnyilicza model's production submodel consisted of two sectors: energy and non-energy. The Hudson-Jorgenson production submodel consists of six sectors:

- Coal
- Crude petroleum
- Refined petroleum products
- Electricity
- Natural gas (crude)
- Delivered gas

* The author is grateful to Dr. Edward Hudson for a helpful discussion.

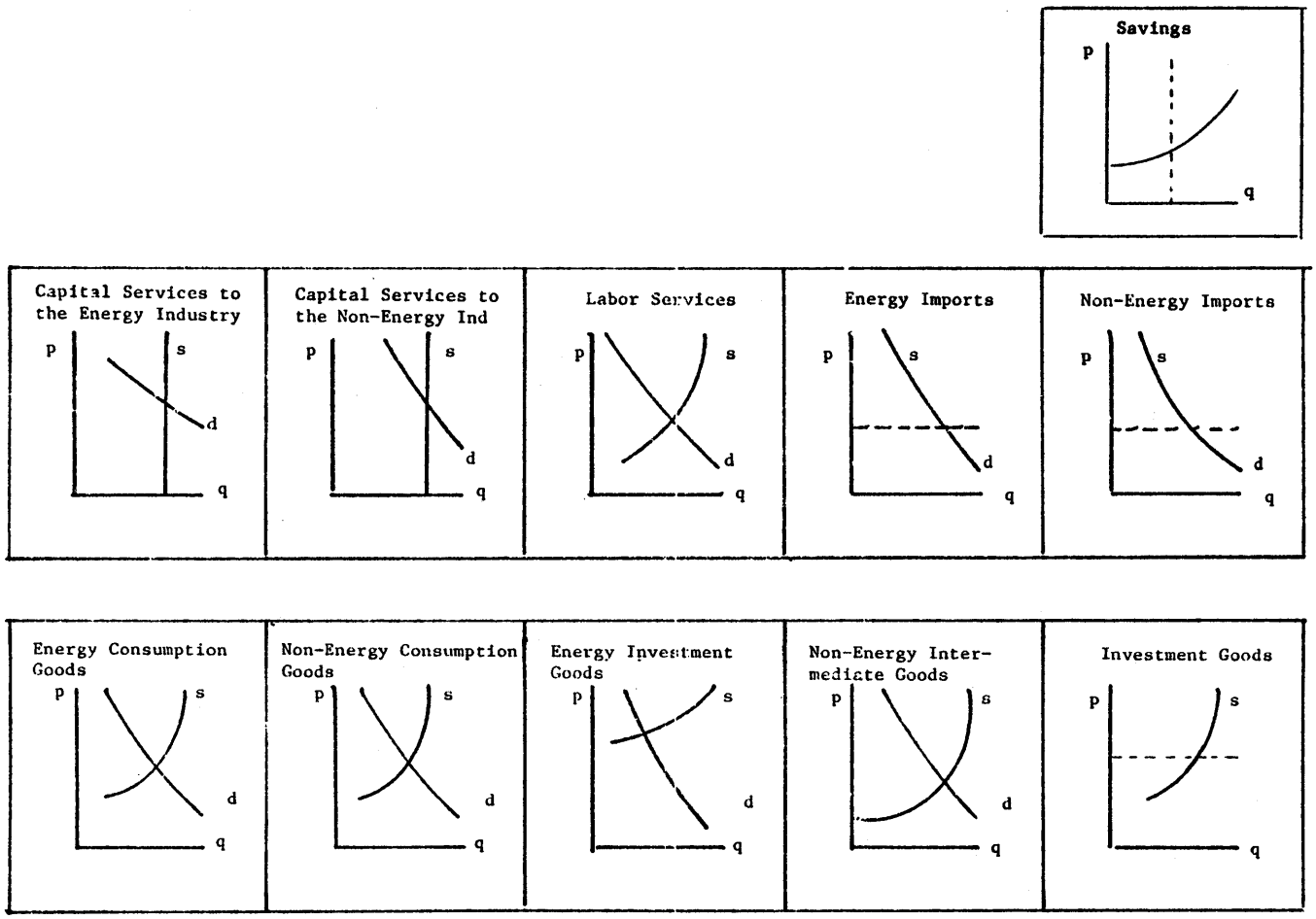


Figure 5—Summary of markets in the Hnyilicza model

THE SRI NATIONAL ENERGY MODEL

Introduction

The SRI National Energy Model is an economic equilibrium model of the U.S. energy system covering all major fuels from primary resources to end use energy consumption (i.e., energy service demands) over the period 1977 to 2025. The model is regionally disaggregated with nine demand regions (such as New England), twenty primary resource supply regions (such as Alaska North Slope), and 600 transportation and distribution links. Conversion technologies such as coal gasification, crude oil refining, and electric power generation by type of fuel are explicitly modeled. The model computes regional prices and quantities of all energy forms, as well as energy technology requirements over the period of 1977 to 2025 by balancing supply and demand.

The SRI model differs in several important respects from the models considered in the previous section. To begin with, the Hudson-Jorgenson and Hnyilicza models are highly aggregated models of the overall economy. In contrast to this, the SRI model is a very disaggregated and detailed model of the energy sector of the U.S. economy. All this detail can be represented graphically by means of a "spatial network." We shall discuss this network just below. Above and beyond this, the SRI model makes use of a different methodology. While the model is an economic equilibrium model in the sense that it determines prices and quantities which are in supply-demand balance,—just as the other models do—it does not make use of the "translog" methodology. There are no variable input-output coefficients. Rather, economic equilibrium is characterized in a much more direct manner. One solves *directly* for prices and quantities which are in balance. Moreover, we can visualize "equilibrium" in this model in terms of a supply-demand balance *at each point* in the spatial network referred to above.

Partly for this reason, we shall start off by discussing the spatial energy network which lies at the heart of the model. Next we shall describe what is meant by "equilibrium" in this context. Finally, we shall gain a deep understanding of the model by analyzing the algorithm used to achieve a supply-demand balance throughout the network.

A spatial network representation of the energy market

The essence of the SRI model is a spatial network representation of the energy system. In the network, the major elements of the U.S. energy system—from the primary resource supplies through the end use energy demands—are described in relationship with each other. A sample energy network is shown in Figure 6. In this figure, the resource supply curves are at the bottom; the usable energy demand curves are at the top. Between these curves is the network describing the entire energy system. The current network has about 2400 materials, processes, and transportation links. A *material* is a primary resource product, or usable energy form *at a specific location*. A material is represented by a *node* in Figure 6. A *process* represents a sector of the energy industry such as coal mining or gasification at a specific location or a class of consumers using a particular energy-consuming device. A *transportation link* represents the process of moving a material from one location to another.

To get a sense of the many paths in the network, consider first the path where coal is mined, converted into synthetic (high Btu) gas, piped to a demand center in a demand region, distributed to industrial users, and consumed as boiler fuel to produce steam. The same end use market could be supplied by coal transported by unit train, distributed to the same industrial users, and used in a coal boiler to produce steam. These two paths can be traced in Figure 6. In the SRI National Energy Model, there are 24 end uses (such as industrial steam) in each of the nine demand regions, and thirty primary resource supplies (such as coal) in the various resource basins. The alternative conversion technologies in the model include all important types of electric power generation (in base, intermediate, and peak load applications), crude oil refining, coal gasification, methanol from coal, hydrogen production from coal, transportation, distribution, and end use conversion.

The model explicitly considers the evolution of the energy system through time—it calculates prices and quantities in a number of time periods. It is important to recognize that the solution to this "dynamic" model is not equivalent to using a "static" model in each of the time periods. Rather,

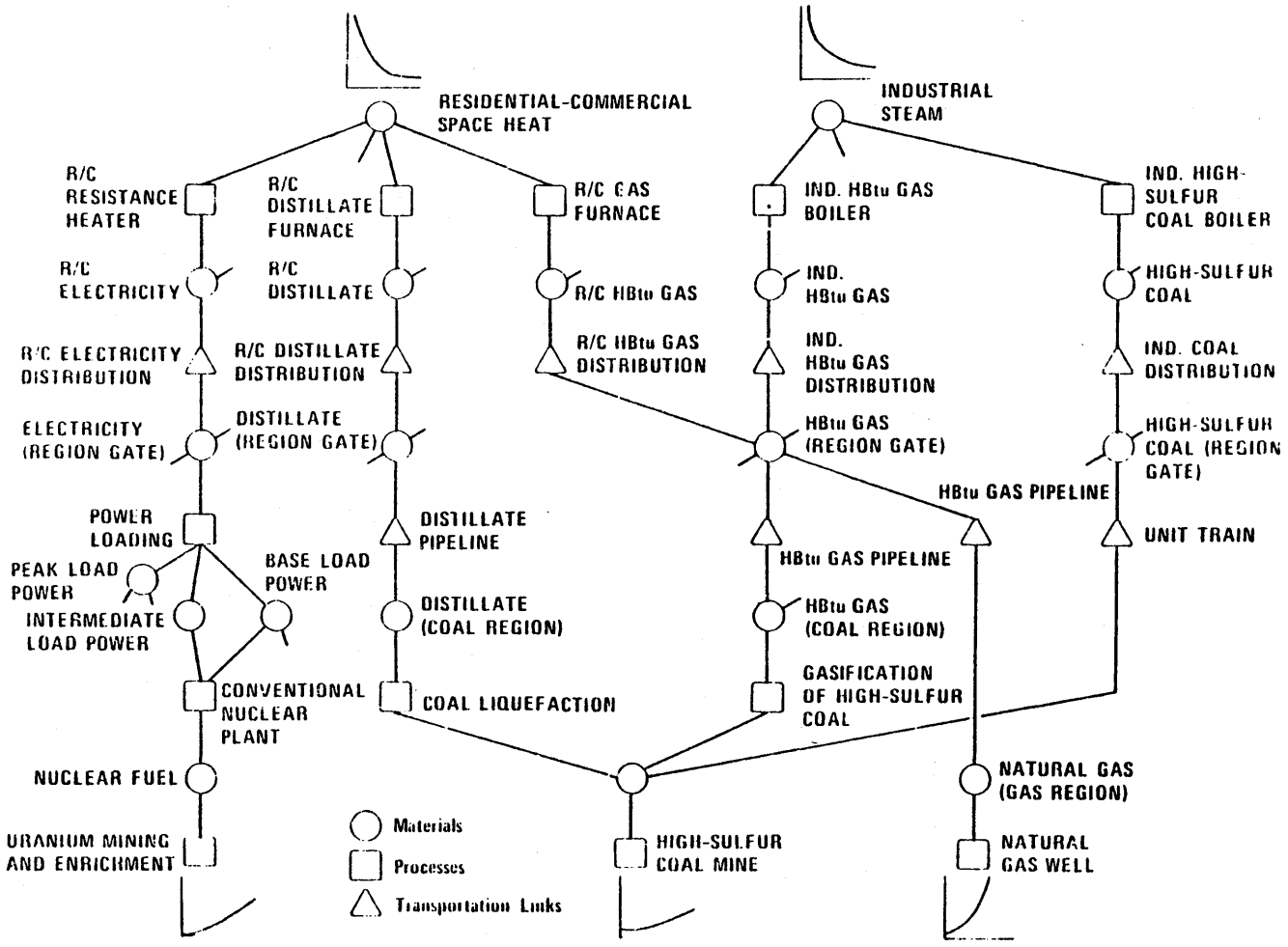


Figure 6—Example of elements of the energy model

the prices and quantities in each period are described by equations interrelating both past and future prices and quantities.

On the meaning of economic equilibrium in the SRI model

In their earlier research, Brock and Nesbitt[3] have drawn upon economic theory to establish the following results concerning economic equilibrium in spatial networks:

1. There exists an *aggregate demand curve* and an *aggregate supply curve* at every *material node* (circle) in the network;
2. There exists a supply curve and a demand curve on every *link* in the network. Any such pair characterizes the supply/demand conditions governing the behavior of the (unique) conversion or transportation process attached to the link.

The *equilibrium prices and quantities* are those for which all *aggregate* supply/demand curve pairs are simultaneously balanced at all material nodes; or *equivalently* for which *all* supply/demand curve pairs are balanced on all the links of the network.

The operation of the model

The SRI model is best described in terms of the algorithm used to solve for equilibrium, and in terms of the relationship of the algorithm to the spatial network and its components. Therefore, in this section we shall describe the algorithm in considerable detail. When we say that this algorithm "solves" the spatial network we simply mean that it determines the set of equilibrium prices and quantities which obtain at each point (link and node) in the network.

The algorithm consists of four essential steps which are repeated at each iteration. The procedure is illustrated at a fairly abstract level in Figure 7. Beginning with an initial "guess" at the equilibrium quantity q_{k-1}, the four steps of the algorithm are:

1. To apply (inverse) resource supply function*

   $p_k = S^{-1}[q_{k-1}]$

2. To proliferate prices up the network (supply function transformation)

   $p_{k+1} = T_s[p_k]$

3. To apply usable energy demand function

   $q_{k+1} = D[p_{k+1}]$

4. To proliferate quantities down the network (demand function transformation)

   $q_{k+2} = T_D[q_{k+1}]$

* An inverse supply function expresses price as a function of quantity.

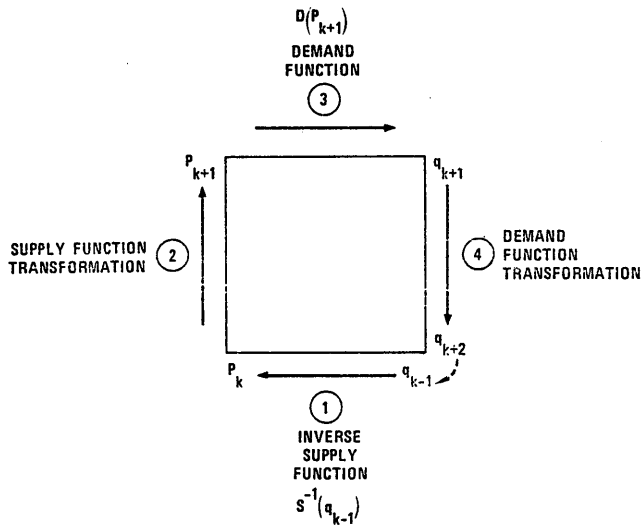


Figure 7—SRI model algorithm

We will now discuss each of these four steps in some detail, and in a less abstract manner.

1. *Inverse Supply Function:* In the first step of the algorithm, the inverse supply function describing each of the primary resources at the *bottom* of the network in Figure 6 is applied. The algorithm begins with a "guess" at the pro-

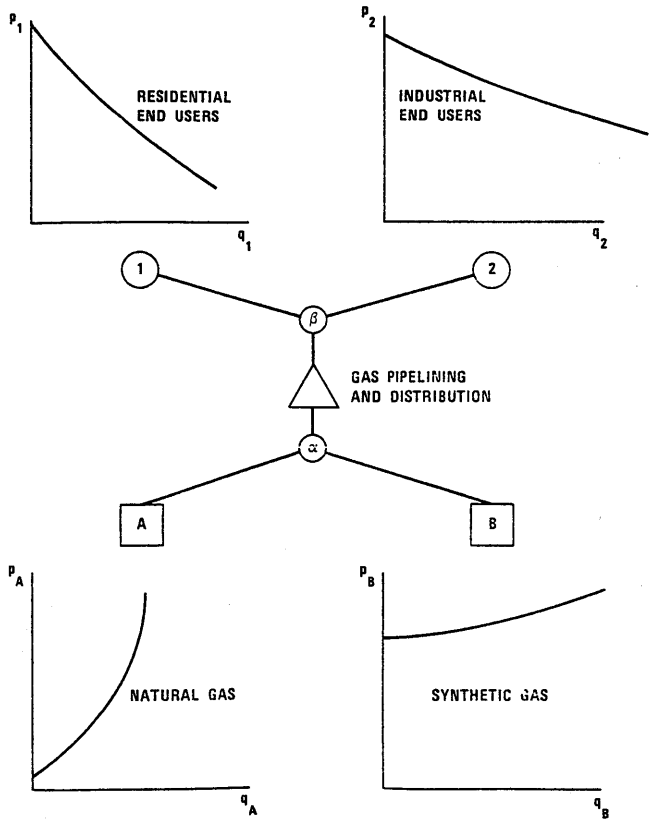


Figure 8—Illustrative gas network

duction level over time for each of the primary resources (denoted $q_{k-1}(t)$ as in Figure 7, where the t denotes a vector over time).

Using the guess $q_{k-1}(t)$, the inverse supply function is applied to give $p_k(t)$, the price of the corresponding primary resource. Schematically, Step 1 is given by

$$p_k(t)=S^{-1}[q_{k-1}(t)]$$

The inverse supply functions are given exogenously, one for each primary resource.

The first step of the algorithm is to guess initial quantities $q_A{}^0$, $q_B{}^0$ of natural gas and synthetic gas respectively and, using the natural gas and synthetic gas supply curves, to calculate the corresponding prices $p_A{}^1$ and $p_B{}^1$ for natural and synthetic gas. These prices apply at the process boxes A and B in Figure 8.

2. *Supply Function Transformation:* In the second step of the algorithm, the intermediate technology submodels are used to compute product prices given input fuel prices.

The technology models begin by assuming that the measure of profitability to a firm which builds a unit of capacity at time t is given by the net present value of the cash flow generated by that unit of capacity:

$$NPV(t)=-c(t)+\sum_{T=t}^{t+L}\frac{p(T)-\phi(T)}{(1+r)^{T-t}}$$

where

$c(t)$=capital cost of the new unit of capacity built at time t;
$p(T)$=product price at time T;
$\phi(T)$=operating cost (including fuel cost) at time T;
r=discount rate (cost of capital)
L=life of technology.

In order to simplify our discussion, we will assume that technologies have a useful life of three years ($L=3$) and that we wish to consider the interrelationship of prices over a three-year time horizon.

It is known from elementary economic theory that at equilibrium, profit maximizing firms earn zero profit. Expressed in terms of net present value, the zero profit assumption is equivalent to:

$$NPV(t)=0 \qquad t=1,2,\ldots$$

where we have assumed that the cost of debt and equity capital are included in the discount rate r. Assuming $NPV(t)=0$ for new plants built in Years 1, 2, and 3, we can write:

$$c(1)=p(1)-\phi(1)+\frac{p(2)-\phi(2)}{1+r}+\frac{p(3)-\phi(3)}{(1+r)^2}$$

$$c(2)=p(2)-\phi(2)+\frac{p(3)-\phi(3)}{1+r}+\frac{p(4)-\phi(4)}{(1+r)^2}$$

$$c(3)=p(3)-\phi(3)+\frac{p(4)-\phi(4)}{1+r}+\frac{p(5)-\phi(5)}{(1-r)^2},$$

which interrelates the equilibrium prices (and, of course, the capital and operating costs and the discount rate). If the firm which is trying to decide whether to add new capacity at time $t=3$ *knew* with certainty what future prices [$p(4)$ and $p(5)$] and future operating costs [$\phi(4)$ and $\phi(5)$] would be, it could, by rearranging the third equation above, calculate the minimum price in Year 3, $p(3)$, below which it would not add capacity and above which it would add more capacity.*

$$p(3)=\phi(3)+c(3)-\underbrace{\frac{p(4)-\phi(4)}{1+r}-\frac{p(5)-\phi(5)}{(1+r)^2}}_{\text{capital charge}}$$

Knowing $p(3)$, $\phi(3)$, $p(4)$, and $\phi(4)$, the second equation of the above three can be used to solve for $p(2)$. A similar calculation gives $p(1)$.

To summarize, for each technology, given the fuel prices (contained in $\phi(t)$), given assumptions about prices and operating costs past the end of the model horizon, and given process economic data ($c(t)$ and $\phi(t)$), product prices are calculated recursively backward in time

$$p(3),\ p(2),\ p(1)$$

as indicated above. When we arrive at the top of the network, we will have calculated the prices $p_{k+1}(t)$ in Figure 7. We have denoted this upward proliferation of prices by using the notation $p_{k+1}(t)=T_s[p_k(t)]$.

Returning to Figure 8, Step 2 of the algorithm involves taking the prices $p_A{}^0$, $p_B{}^0$ (and the quantities $q_A{}^0$, $q_B{}^0$) and determining the corresponding prices $p_1{}^0$, $p_2{}^0$ at the top of the network. The first step is to compute the average price of gas entering the pipeline at α in Figure 8 as follows:

$$\bar{p}=\frac{p_A{}^0q_A{}^0+p_B{}^0q_B{}^0}{q_A{}^0+q_B{}^0}$$

The price of gas delivered to the two end users (i.e., to β in the figure) and would simply be

$$p_1{}^0=p_2{}^0=\frac{\bar{P}}{e}+c.$$

3. *Demand Function:* The third step in the algorithm is to apply the direct demand function at the top of the network using the prices $p_{k+1}(t)$ computed on the way up in Step 2. That is, an estimate of demand is obtained by applying the demand function

$$q_{k+1}(t)=D[p_{k+1}(t)].$$

Returning to Figure 8, this step of the algorithm involves taking the price $p_1{}^0$ of residential gas determined by Step 2 and finding the residential demand $q_1{}^0$ at that price using the residential demand curve. A similar procedure yields the industrial demand $q_2{}^0$ corresponding to the industrial gas price $p_2{}^0$ determined in Step 2.

4. *Demand Function Transformation:* In the fourth step of the algorithm, the technology submodels are used to compute fuel requirements given product demands, and market share submodels are used to divide demand among compet-

* Equilibrium would obtain at the price $p(3)$ itself because the firm would just be indifferent between adding and not adding capacity.

ing technologies based on the price computed for those technologies in Step 3. We will discuss both calculations briefly.

First, each technology is characterized by a thermal efficiency $e(t)$ for new units built at time t and thus by an average efficiency $\bar{e}(t)$ for all plants in place at time t. Hence, the fuel required is given by

$$q_{k+2}(t)=q_{k+1}(t)/\bar{e}(t).$$

Second, if the quantity $q_{k+2}(t)$ is to be split among a number of competing suppliers, a "market share" model is applied to the quantity $q_{k+1}(t)$ to split it among competing supply technologies based on the prices computed for those technologies in Step 2. We do not have space enough to discuss this market split model here.

This step of the algorithm is better explained in terms of the network in Figure 8. Referring to that figure, the fourth step of the algorithm actually consists of two operations. First, the residential and industrial quantities $q_1{}^0$ and $q_2{}^0$ respectively determined in Step 3 are added to give the total demand for gas $\bar{q}=q_1{}^0+q_2{}^0$ at β in the figure. In order for the pipeline to deliver $\bar{q}$ units of gas at β, it must take in $\bar{q}/e$ units of gas at α, where e is the thermal efficiency of the pipeline, including transportation losses.

Given the demand $\bar{q}/e$ of gas at α, we then apply the market share model to compute how much of that gas would be synthetic and how much would be natural at the prices $p_A{}^0$, $p_B{}^0$ determined in Step 2. That is, we would compute

$$q_A{}^1=MS_A(p_A{}^0, p_B{}^0)\times\bar{q}/e$$

$$q_B{}^1=MS_B(p_A{}^0, p_B{}^0)\times\bar{q}/e$$

where $MS_A(p_A{}^0, p_B{}^0)$ is the fraction of the demand that would be satisfied by Process A (natural gas) if the prices of natural and synthetic gas were $p_A{}^0$ and $p_B{}^0$ respectively, and similarly for Process B.

Characterization of equilibrium in the network

If $q_A{}^1\cong q_A{}^0$ and $q_B{}^1\cong q_B{}^0$ equilibrium has been achieved (recall that $q_A{}^0$ and $q_B{}^0$ were our initial guesses in Step 1). If not, we return to Step 1 with the new guesses $q_A{}^1$ and $q_B{}^1$, and repeat the four steps, as many times as necessary to achieve equilibrium.

THE ROLE OF ECONOMIC MODELS IN NORMATIVE POLICY ANALYSIS

We shall conclude the present essay by issuing some precautionary remarks about the role of economic models in normative public policy analyses. The interested reader will find an elaboration of these remarks in Chapter I of Brock and Nesbitt.[3]

It must be appreciated that the models discussed in this report are economic in nature. That is, their aim is to project *prices* and *quantities* of energy (and, sometimes, non-energy) commodities. Yet normative policy analysis is interested not only in the *economic* dimension of alternative policies, but also in the *political* and the *ethical* dimensions.

For example, a policy maker might well wish to have at least probabilistic information about the likely price/quantity impacts of pursuing alternative strategies. For an example of this, see the companion paper presented by Dr. Steven N. Tani. Tani points out how price quantity data generated by an energy model can be used to make economic welfare judgments about alternative policies by means of the criterion of "net social surplus." This is important—indeed, it is far the most important role of strictly economic models in *normative* policy analyses.

But the same policy maker might also wish to know the political implications of pursuing alternative strategies. For example, a policy which looks good on economic grounds may for various reasons be "politically unacceptable." Game theoretic models can in principle be used to determine which policies are politically acceptable in a balance-of-power sense. But economic equilibrium models are not game theoretic models, and should not be confused as such.

Finally, there are the distributional concerns which are central to normative ethics. Economic planning models in and of themselves cannot furnish us with the answer as to which policy is most "fair." President Carter has repeatedly emphasized his concern with equity. But in this matter we have a problem for ethics—not for price/quantity models.

While these points may seem too obvious to need explicit statement there is often confusion about the proper role of economic data in normative policy analysis. For this reason, we felt it worthwhile to close with a reminder. There is one other important limitation to economic models of the kind reviewed above. This concerns the use of an "equilibrium" methodology in the context of regulated commodities such as natural gas. Many problems of interpretation arise here; but they are outside the scope of the present paper.

REFERENCES

1. Hudson, Edward and Dale W. Jorgenson, "U.S. Energy Policy and Economic Growth, 1975-2000," *Bell Journal of Economics and Management Science, 1974, Volume 5 No. 2.*
2. Hnyilicza, Esteban, "An Aggregate Model of Energy and Economic Growth," M.I.T. Energy Laboratory Working Paper MIT-EL-75-010WP, 1975.
3. Brock, Horace W. and Dale Nesbitt, "Large-Scale Energy Planning Models: A Methodological Analysis," Report prepared for the Office of Policy Research and Analysis of the National Science Foundation, May 1977.

Decision analysis of the synthetic fuels commercialization program

by STEVEN N. TANI
SRI International
Menlo Park, California

INTRODUCTION

In his State of the Union Message in January 1975, President Gerald Ford called for the accelerated development of U.S. energy technology and resources and proposed a comprehensive set of energy supply and conservation measures to reduce U.S. requirements for imported oil. As one of these measures, the President proposed a federal incentive program whose goal would be the commercial production of one million barrels per day of synthetic fuels by 1985. In such a program, the Federal Government would provide suitable financial and regulatory incentives to stimulate private sector investment in commercial-scale plants to convert coal, oil shale, and other relatively abundant domestic resources into clean liquid and gaseous fuels. It was generally believed that without such incentives, industry would be unlikely to undertake the large risks of synthetic fuel plant investments.

The benefits to be achieved by the synthetic fuels program would be the following:

1. An accelerated accumulation of experience and information on the technical, environmental, economic, and institutional aspects of commercial-scale synthetic fuel production for better-informed private sector investment decisions.
2. The development of an industry infrastructure to support subsequent expansion of the synthetic fuels industry.
3. Insurance against high world oil prices and against early depletion of domestic sources of conventional fuels.
4. Protection against the losses of an oil embargo.
5. Improvement in the U.S. international bargaining position.

These benefits, however, would be counterbalanced by the possible costs of subsidizing synthetic fuels relative to less expensive energy sources such as imported oil and other domestic resources and by the environmental and socioeconomic costs associated with rapid development of coal and oil shale reserves.

In response to the President's message, the Interagency Task Force on Synthetic Fuel Commercialization was formed to evaluate the economic and environmental costs and benefits of the program and to recommend to the President a program of appropriate size and scope. The Task Force was chaired by the Office of Management and Budget and included members from the Federal Energy Administration, the Environmental Protection Agency, the Departments of State, Commerce, and Treasury, the Council on Environmental Quality, and the National Science Foundation. We, in the Decision Analysis Group at Stanford Research Institute (now SRI International), were engaged to assist the Task Force in conducting an analysis of the program.

STRUCTURE OF THE ANALYSIS

The fundamental question addressed by this analysis was whether the U.S. should have a synthetic fuels commercialization program and, if so, how large the program should be. The Task Force defined four distinct program alternatives to be evaluated:

1. No Program—No federal funding of synthetic fuels commercialization but continuation of research and development.
2. Informational Program—A minimal program designed primarily to generate technical, environmental, and economic data on various resource-to-fuel conversion processes, with synthetic fuel production of about 350,000 barrels per day by 1985.
3. Medium Program—A program designed to generate more complete information on a wider range of processes and to meet the President's goal of 1,000,000 barrels per day by 1985.
4. Maximum Program—A program designed to achieve the greatest amount of synthetic fuel production in 1985 possible without causing major dislocations in the economy: 1,700,000 barrels per day.

The object of the analysis was to determine which of these alternatives would be of greatest net benefit to the nation as a whole, where net benefit includes both economic and noneconomic impacts.

We defined four components of net national benefit for the evaluation of the program alternatives: economic impact on consumers, economic impact on producers, embargo protection, and environmental and socio-economic impacts.

To measure the economic impact on consumers, we utilized the concept of consumer surplus. Consumer surplus is the difference between the value of a good to consumers and the amount of money they must pay for it. This is shown graphically in Figure 1. The demand curve, by definition, is the most consumers would pay for each unit of the good, which is the value of that unit. If the market price is p, then q units will be purchased. For every unit except the last one, the value of the good exceeds the price paid for it. The shaded area between the price line and the demand curve represents the total excess value the consumers receive from this good; this is called the consumer surplus.

In the case of the synthetic fuels program, it was felt that a demonstration that synthetic fuels could be produced cheaply, if achieved, would have the effect of holding down the price of imported oil. The resulting increase in consumer surplus would then be credited as a positive benefit of the program.

To measure the impact on producers, we used a concept analogous to that of consumer surplus—producer surplus. This is the difference between the amount producers receive for a good and their marginal cost of producing it. Clearly, producer surplus is directly related to the idea of profitability. Figure 2 shows producer surplus graphically. The supply curve represents the marginal cost of producing each unit of the good, which is the least amount of money the producers would accept for it. The shaded area between the price line and the supply curve is equal to the total producer surplus for that good.

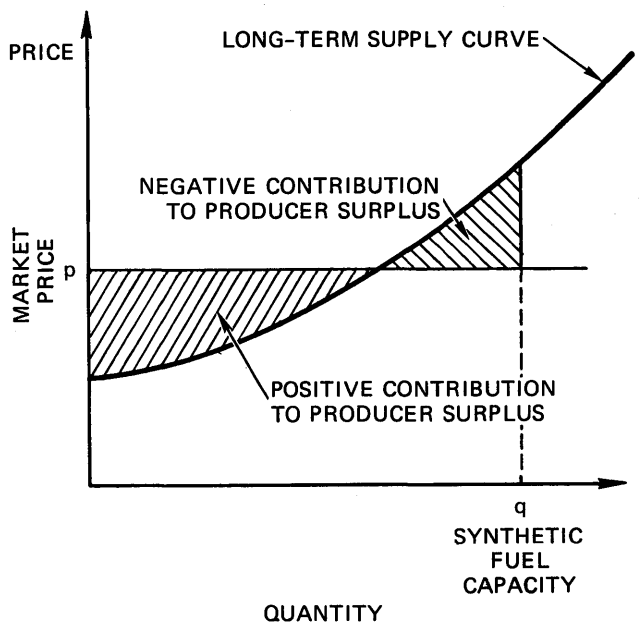


Figure 2—Producer surplus

We assumed in this analysis that synthetic fuel would be a substitute for imported oil. Therefore, if the cost of the synthetic fuels turned out to be less than the cost of imported oil, the industry would accrue positive producer surplus, which would be credited to the program as a benefit. How-

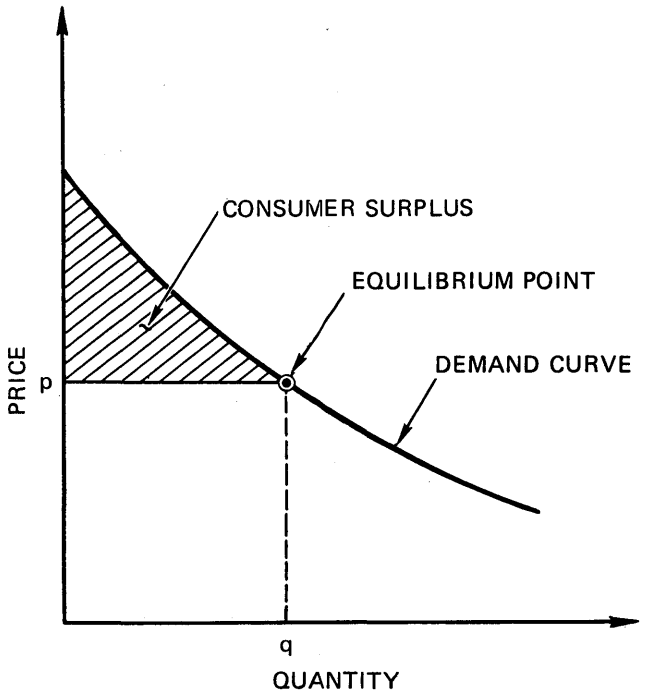


Figure 1—Consumer surplus

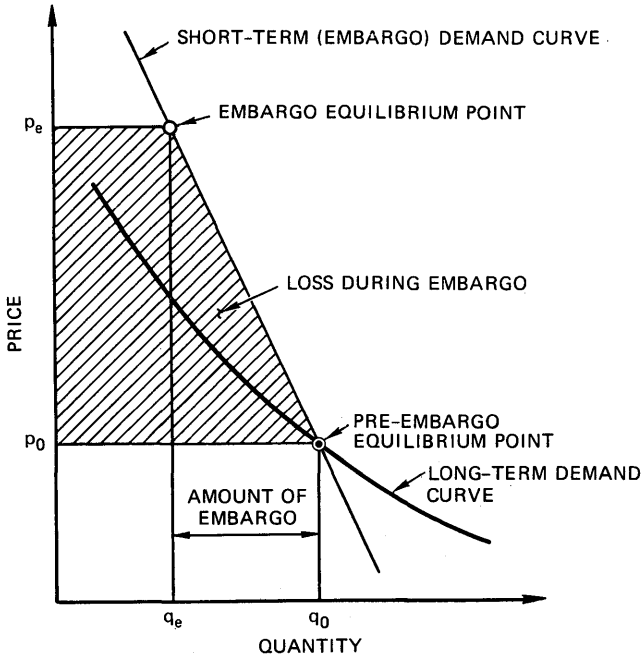


Figure 3—Embargo loss

ever, if synthetic fuels turned out to be costlier than imported oil, producer surplus would be negative and industry would require a subsidy from the government to cover its losses. The amount of this negative producer surplus would be charged as a cost of the program.

The algebraic sum of consumer and producer surplus is a measure of the total economic impact of the program on the nation assuming normal market conditions. However, it does not include the impact of the program in the event of an oil embargo. The situation during an oil embargo is illustrated in Figure 3. The pre-embargo price and quantity of oil are established on the long-term demand curve. If an embargo occurs, the quantity of oil available for consumption decreases abruptly. Because of short-term inflexibilities in consumption patterns, the marginal value (or shadow price) of oil is much higher than the long-term demand curve indicates. Here we use a linear short-term demand curve to show this effect. The economic cost of the embargo is the loss of consumer surplus during the embargo and is represented by the shaded trapezoidal area.

The synthetic fuels program, by providing a substitute for some of the imported oil, would reduce this embargo loss by increasing the amount of fuel available for consumption during the embargo. This reduction of embargo loss, weighted by the probability of occurrence of an embargo, is credited to the program as a benefit.

Finally, the synthetic fuels program would result in non-economic costs in the form of environmental damage (e.g., increased air pollution) and socio-economic disruption (e.g., "boom towns" near mining and conversion facilities). These costs, to the extent that they are not internalized in the producers' costs (e.g., pollution control costs), are charged to the program.

The sum of these four components of program impact is the measure of net national benefit we used in the analysis to evaluate the alternatives.

Clearly, to determine the net benefit of each program alternative, we need to know something about the energy supply and demand situation in the future. There was, of course, considerable uncertainty about the future energy picture, so we used probabilistic modeling techniques to quantify and incorporate the uncertainty in the analysis.

Figure 4 shows the decision tree structure that we ultimately developed for this analysis. We treated the dynamics in a simple manner by looking at three discrete time periods. In 1975, the government would make its program decision,

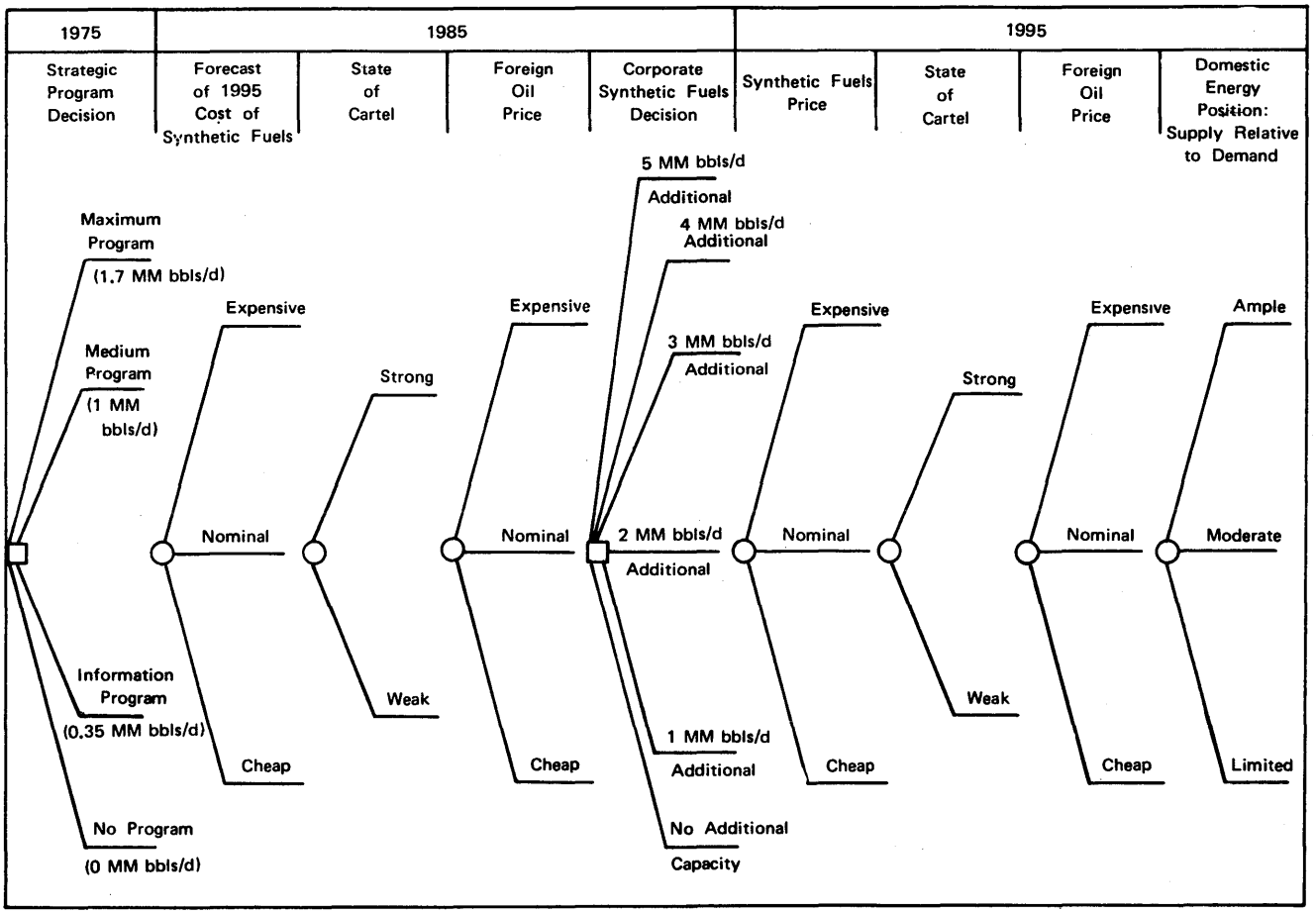


Figure 4—Decision tree

choosing one of the four alternatives. Then, in the mid-1980's, the program would result in information about the ultimate cost of synthetic fuels production. Based on this information and on the prevailing and projected price of imported oil, the industry would make its decision on further investment in synthetic fuel plants. The price of imported oil would, of course, depend on whether or not the oil producers' cartel remained effective in controlling prices. Finally, in the mid-1990's, when the new synthetic fuel plants are on-stream, the program impacts can be determined by looking at the cost of synthetic fuels, the price of imported oil, which again depends on the current state of the cartel, and the U.S. energy supply and demand balance.

The year 1985 was used to typify the decade of the 1980's and the year 1995 to typify the decade of the 1990's. Program cost and benefits were measured in constant 1975 dollars and were discounted in 1975 using a discount rate of ten percent.

The decision tree in Figure 4 shows how uncertainty was explicitly incorporated in the analysis. Uncertainty about each of the factors shown was quantified in the form of probabilities. Then, for each combination of factors, which defined a unique scenario of the future, both the probability of occurrence for that scenario and the discounted net national benefit associated with it were calculated. Finally, for each alternative, the *expected* net benefit was calculated by weighting the outcome of each scenario by its probability and summing. Note that the decision tree in Figure 4 defines 5,832 different scenarios for each of the four program alternatives.

The industry decision in 1985 of how much further investment to make in synthetic fuels plants required special treatment. While the government decision would be made on the basis of overall national benefits, the private sector decision would be made on the basis of corporate profits only. Therefore, in the analysis, the level of corporate investment that maximized expected future producers surplus was selected.

Figure 5 illustrates the techniques we used to quickly

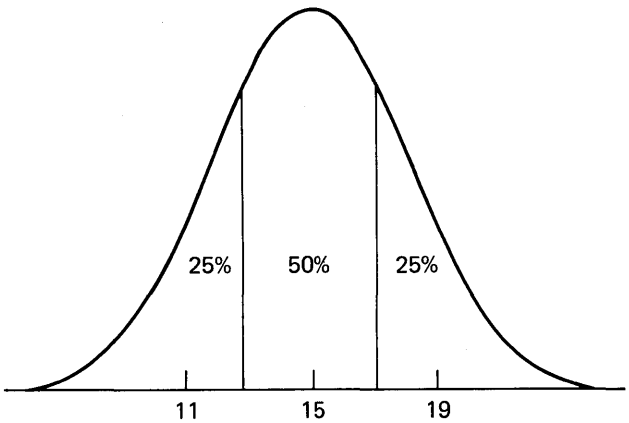


Figure 5—Simple encoding technique

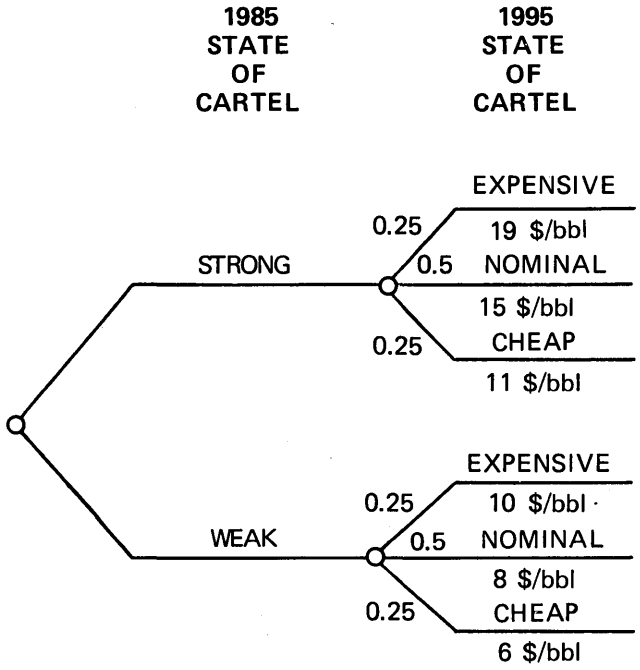


Figure 6—Probabilities of 1985 imported oil prices

encode uncertainty in the various factors. The probability distribution shown represents uncertainty in the 1985 imported oil price given that the cartel is strong. According to this distribution, it is equally likely that the price will be above or below $15 per barrel (the median value). Also, there is a ten percent chance that the price will be below

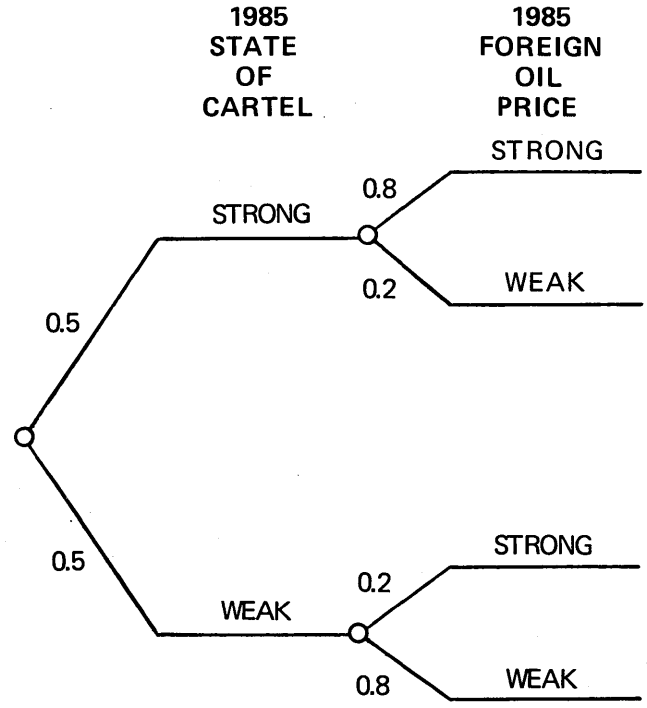


Figure 7—Probabilities of state of cartel

$11 per barrel (the 10 percent fractile) and a ten percent chance that it will be above $19 per barrel (the 90 percent fractile). We divided the distribution into three sections having areas of 25 percent, 50 percent, and 25 percent and used the median value to represent the middle section and the 10 percent and 90 percent fractiles to represent the two tails. Thus, as shown in Figure 6, we said in the analysis that there is a 25 percent chance that the 1985 imported oil price would be $19 per barrel, a 50 percent chance that it would be $15 per barrel, and 25 percent that it would be $11 per barrel, given that the cartel is strong. The imported oil price given that the cartel is weak is assessed to be much lower—$10, $8 and $6 per barrel, respectively.

Using this simple technique, we encoded the uncertainty of the Task Force members about each of the factors shown in the decision tree. Of particular interest are the assessments of the future state of the oil producers' cartel. As shown in Figure 7, the chances of the cartel remaining strong through 1985 were assessed by the Task Force to be 50-50. Given that it is strong in 1985, the probability that it would remain strong through 1995 was assessed to be 80 percent, while if it is weak in 1985, the chance that it would become strong by 1995 was assessed to be only 20 percent.

RESULTS OF THE ANALYSIS

After we had structured the problem with the decision tree, constructed a computer model to calculate the net national benefit for each of the thousands of scenarios in the tree, and encoded the uncertainties of the Task Force, we were ready to compute the analytic results.

Figure 8 summarizes these results. The total expected discounted net benefit (in billions of 1975 dollars) is shown, along with its components, for each of the three synthetic fuels program levels relative to having no program at all. These results indicated that, on balance, the synthetic fuels commercialization program was not in the best national interest and that the bigger the program, the greater the national loss. The small informational program had an expected impact on minus $1.65 billion. The larger program had expected impacts of minus $5.41 billion and minus $10.98 billion, respectively.

We can get more insight by looking at the components of total net benefit. While the synthetic fuels program is expected to have positive impacts on consumer surplus

	Expected Discounted Net Benefit (billions of 1975 dollars)				
Program Alternative	Consumer Surplus	Producer Surplus	Embargo Protection	Environmental and Socioeconomic	Total
No Program	0	0	0	0	0
Information Program (0.35 mm bbl/day)	1.07	-2.71	0.43	0.44	-1.65
Medium Program (1 mm bbl/day)	3.29	-8.74	1.18	-1.14	-5.41
Maximum Program (1.7 mm bbl/day)	4.55	-15.77	2.23	-1.99	-10.98

Figure 8—Expected program impacts

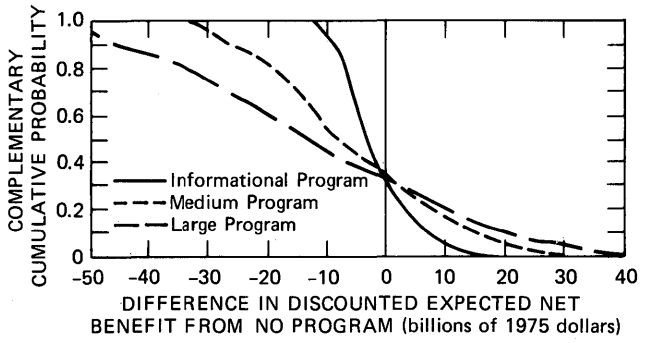


Figure 9—Uncertainty in program impacts

through the possible moderation of future imported oil prices and on embargo losses through a slight reduction in oil imports, these benefits are far outweighed by the negative impact on producer surplus. Basically, it was far more likely than not that synthetic fuels would be more expensive than imported oil and therefore need a subsidy. The negative impact of environmental and socio-economic costs is relatively minor.

The results shown in Figure 8 are the expected values of program impacts. There is, of course, considerable uncertainty about the impact of the program, as shown in Figure 9. While the expected impact of the informational program is $1.65 billion, there is a 30 percent chance that the net impact will be positive and a 10 percent chance that it will be as much as +$7 billion. On the other hand, there is a 10 percent chance that it will be as negative as −$9 billion. It is equally likely that the impact will be worse than or better than −$4 billion. The uncertainty in the impact of the larger program is even greater.

Figure 10 is a partial expansion of the decision tree that shows how two of the factors affect the results of the analysis. The −$1.65 billion expected impact of the information program consists of a 50 percent chance of −$4.86 billion if the cartel in 1985 is weak and a 50 percent chance of +$1.55 billion if it is strong. Note that a weak cartel, which leads to generally lower imported oil prices, is bad for the syn-

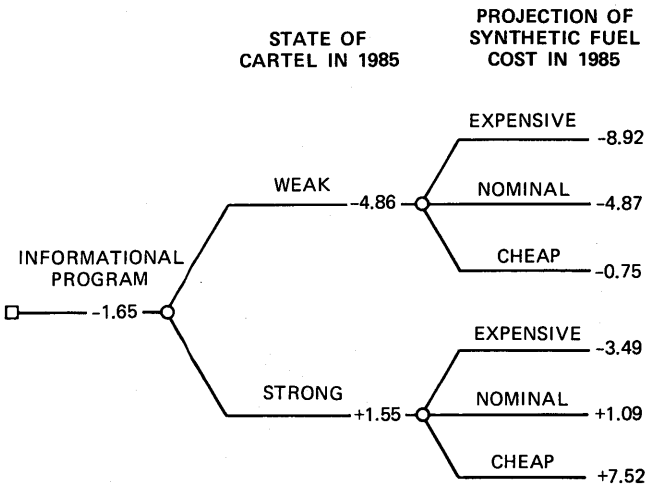


Figure 10—Partial expansion of tree

thetic fuels program while presumably very good for the nation as a whole. Conversely, a strong cartel, with higher imported oil prices, makes the program look good while being bad for the nation. This emphasizes that the synthetic fuels program is a hedging strategy—it pays off when other things are going badly. Note also that if the cartel is weak, the program looks bad even if synthetic fuels turn out to be cheap to produce. On the other hand, if synthetic fuels turn out to be expensive, the program looks bad even if the cartel is strong. That is why, on balance, the program looks bad.

The assessment of a 50-50 chance that the cartel would remain strong through 1985 turned out to be pivotal and more than a little controversial within the Task Force. To show the implications of different probabilities for this factor, we performed a sensitivity analysis, which is shown in Figure 11. This gives the expected net impact of each program level relative to no program as a function of the probability of a strong cartel in 1985. It assumes that with 80 percent probability, the cartel will remain in the same state from 1985 to 1995. Figure 11 shows that only if the probability of a strong cartel in 1985 exceeds 75 percent does the information program look better than no program and that the probability must exceed 82 percent for the medium size program to be the best alternative. An interesting result is that the maximum size program is never optimal for any value of this probability.

So far, the analytic results have been presented only in terms of expected values. It might be argued that the decision should not be made on the basis of expected values but rather on the basis of values that are adjusted for risk. To show how various levels of risk aversion would affect the

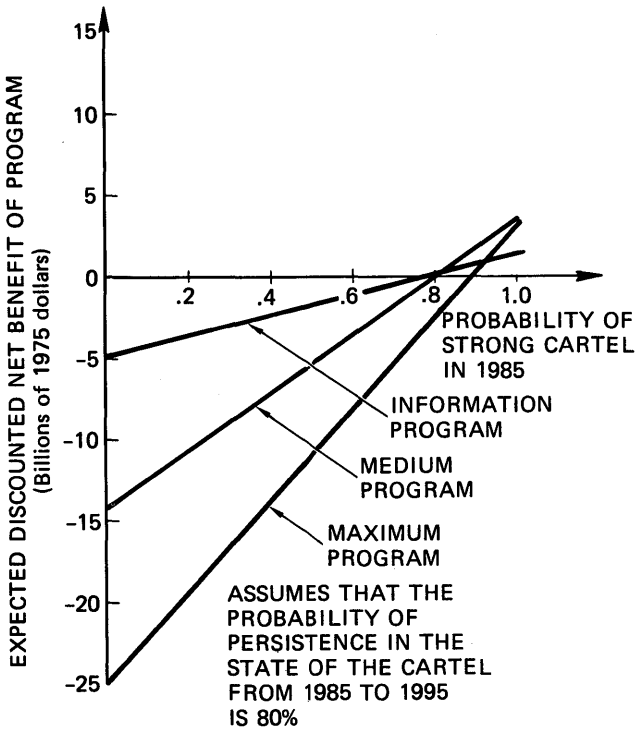


Figure 11—Sensitivity to cartel probabilities

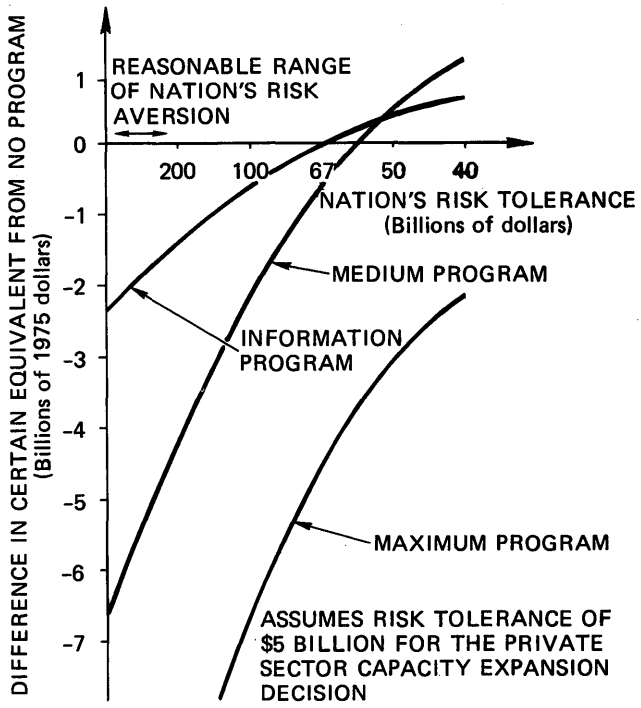


Figure 12—Sensitivity to nation's risk aversion

results, we prepared the sensitivity profile shown in Figure 12. Here, we have assumed that the nation's risk attitude is expressed by one of the family of exponential utility curves. The degree of risk aversion expressed by this curve is given by its one parameter, called the risk tolerance; the smaller the risk tolerance, the greater the degree of risk aversion. In personal terms, an individual's risk tolerance is the largest amount of money he would willingly risk in a gamble that is equally likely to halve or double that amount.

Figure 12 shows the value to the nation of each program level relative to no program as a function of the nation's risk tolerance assuming an industry risk tolerance of $5 billion for the private sector capacity expansion decision. Note first that the value of the program increases as the nation's risk aversion increases. This is characteristic of a hedging strategy, since it reduces overall uncertainty. However, the nation's risk tolerance must be less than $67 billion for the information program to be better than no program and it must be less than $56 billion for the medium size program to be the best alternative. We believe that a reasonable range for the nation's risk tolerance is from one-fourth to one-half of annual GNP, or about $300 billion to $600 billion. As Figure 12 shows, for any risk tolerance in this range, the ranking of program alternatives is the same as in the expected value case, with the best alternative being no program at all.

EPILOGUE

As far as we can determine, this is the first decision analysis to be presented in the White House. The chairman

of the Task Force presented it to the President's Energy Resources Council in July 1975. Citing benefits of the program that were not quantified in the analysis, such as the international leverage gained by the U.S. in asserting positive leadership in developing alternate fuel sources, as well as the "relatively small risk and expected cost" of the small program, the Task Force recommended that the government undertake the informational program alternative with a possibility that it could switch to the medium size program pending additional information on crucial factors. The Administration's bill incorporating this recommendation ultimately failed to pass through Congress.

COMPUTER UTILIZATION

For this analysis, we wrote a FORTRAN program for use on a commercial time sharing system. The program, which we created from scratch rather than use off-the-shelf routines, required approximately 400 lines of code and cost roughly fifteen dollars to run a complete evaluation of the decision tree.

One feature of the time sharing service that we found to be especially useful was the accessibility from different locations. We did most of the model development at SRI headquarters in Menlo Park, California, but we used the program extensively while working with the Task Force in Washington D.C. Indeed, one of the most valuable aspects of our assistance to the Task Force was our ability to answer almost immediately their many questions about the effect of changes on the assumptions and assessments in the analysis.

REFERENCE

Recommendations for a Synthetic Fuels Commercialization Program, report submitted by Synfuels Interagency Task Force to The President's Energy Resources Council, November 1975. Volume I: Overview Report. Volume II: Cost/Benefit Analysis of Alternate Production Levels.

A computer model for examining regional peak electric load growth

by P. F. SCHWEIZER
Westinghouse R&D Center
Pittsburgh, Pennsylvania

INTRODUCTION

One of the critical factors affecting the electric utility and electrical manufacturing industries is the forecast of the long-term demand for electricity. Superficial forecasts based on either past time trends or very broad based factors for the entire United States do not provide sufficient confidence for long-range planning. This effort is an attempt to examine the basic factors in the U. S. which create the demand for electricity. Equally, or possibly even more important, is the examination of the sensitivity of the forecast to the basic driving factors. Through this sensitivity analysis, which relates the changes in input assumptions to output changes, the reasonableness of other forecasts can be established.

POWER SYSTEMS MARKET FORECAST MODEL

When one begins the development of a market forecast model, some important basic concepts must be resolved.[1] The time span of interest in the forecast must be specified. Whether the span is hours, days, months, years, or decades will influence the structure selected for the model. In general, for short time spans, the model structure can be closely patterned after the existing market. For long-term models, however, the dynamics of a changing market must be represented in the model.

Because of different climates, industries, economics, and traditions, geography plays an important role in an electric demand forecast model. The differing regional saturations of electric air conditioning and space heating, mixes of industry, and price of electricity compared with competing fuels require that geographical considerations be included to obtain better forecasts.

Nine data bases are presently maintained for each of the nine NERC regions. These regions are geographically located as shown in Figure 1. One obviously questions the selection of these regions versus some other delineation. When dealing with demand data, such as population, households, value added, etc., a state or census region division might seem more reasonable. However, when dealing with electric supply data, such as generation capacity, additions, costs, etc., an electric utility boundary or power pool area seems reasonable. The basic problem is that data collection for supply and demand is on a completely different geographic basis. Each NERC region is a grouping of Power Supply Areas which are, in turn, a grouping of utilities, many of which overlap state and political boundaries. To establish consistency in the data base, we decided that demand data given at the state level could be more easily transformed to a supply area, such as an NERC region, rather than vice versa. The NERC supply region was selected rather than the power supply areas or utility service areas simply because the model detail we were requiring would need excessive data storage and manipulation at the more disaggregated level.

A block diagram of the model is shown in Figure 2. Submodels for residential, industrial, and commercial are used to estimate kilowatthour (KWH) sales. Exogenous estimates of transportation, agricultural, governmental, and miscellaneous KWH sales are summed with residential, commercial, and industrial to obtain total sales. Total electrical energy output is calculated using transmission and distribution loss estimates determined from input data specifying losses as a percent of KWH sales. The peak load in each region is calculated from total output using a load factor which is estimated as a function of the mix of KWH sales. The installed capacity in each region is calculated from peak load using either an option that inputs the percentage of generation additions of each type or an option that uses generation capital, fuel price, and investment data to determine the least cost mix of additions. The capacity factor for each unit type is calculated by adjusting the KWH output from all units (capacity factor × installed capacity rating × 8760 hrs.) to be equal to the estimated KWH generation from the model. Input energy to all types of units is calculated using the estimated electrical generation and the heat rate for each type of unit. Additional detail on each of the calculational blocks and models depicted in Figure 2 is shown in Figures 3 through 6 and is discussed in the following.

Residential sector model

A block diagram of the model used to estimate residential KWH sales for each NERC region is shown in Figure 3. By

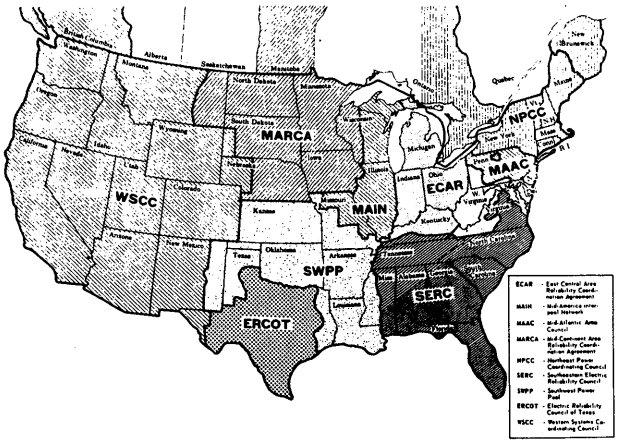


Figure 1—Regional electric reliability councils

multiplying population by headship rate, an estimate of households is obtained. Time trends for the saturation of appliances, air conditioning, and space heating are input to the model[2] and are used to calculate saturations in any given year. These saturations are then adjusted based on price effects and multiplied by the number of households to obtain an estimate of the number of electrical end-use units. Average annual use for each unit type is estimated by inputting time trends of annual use and adjusting annual use based on price effects. An estimate of annual KWH consumption is obtained by multiplying the number of end-use units by the annual use.

The three equations used to estimate residential KWH sales in any given year are:

$$RKWH = \sum_{\text{all appl-}} POP*HDR*SAT \quad (1)$$

$$*SATPA*USE*USEA$$

$$SATPA=\left(\sum_{LOA} RPE/LOA\right)^{\alpha_{SE}}*\left(\sum_{LOA} RPG/LOA\right)^{\alpha_{SG}} \quad (2)$$

$$USEA=RPE^{\alpha_{UE}} \quad (3)$$

where POP is regional population, HDR is regional headship rate, SAT is appliance saturation, SATPA is price adjustment of saturation, USE is annual KWH consumption, USEA is the price adjustments of annual use, RPE and RPG are the relative prices of electricity and gas with respect to the base year, LOA is the lifetime of an average appliance, α_{SE} and α_{SG} are electric and gas price elasticities used to adjust saturation, and α_{UE} is the electric price elasticity used to adjust annual use.

Industrial sector model

The industrial sector model, shown in Figure 4, uses Federal Reserve Board production indices as a national production indicator for two digit manufacturing and mining categories. Projected trends in dollars of value added by SIC code are used to calculate ratios which estimate the share of national production to be allocated to each NERC region in any given year. Input time trends are used to estimate electricity use per unit of production for each SIC code. Industrial KWH consumption is estimated using the regional production estimates and the energy intensiveness estimates corrected for price effects.

The equations used to estimate industrial KWH consumption are given by:

$$INDKWH=\sum_{SIC} FRB_{NAT}*(\$VAREGSIC/\$VANATSIC)$$

$$*USESIC*IUSEA_t \quad (4)$$

$$IUSEA_t=(1-\lambda)IUSEA_{t-1}+RPEI^{\alpha_{UIE}}RPOI^{\alpha_{UIG}} \quad (5)$$

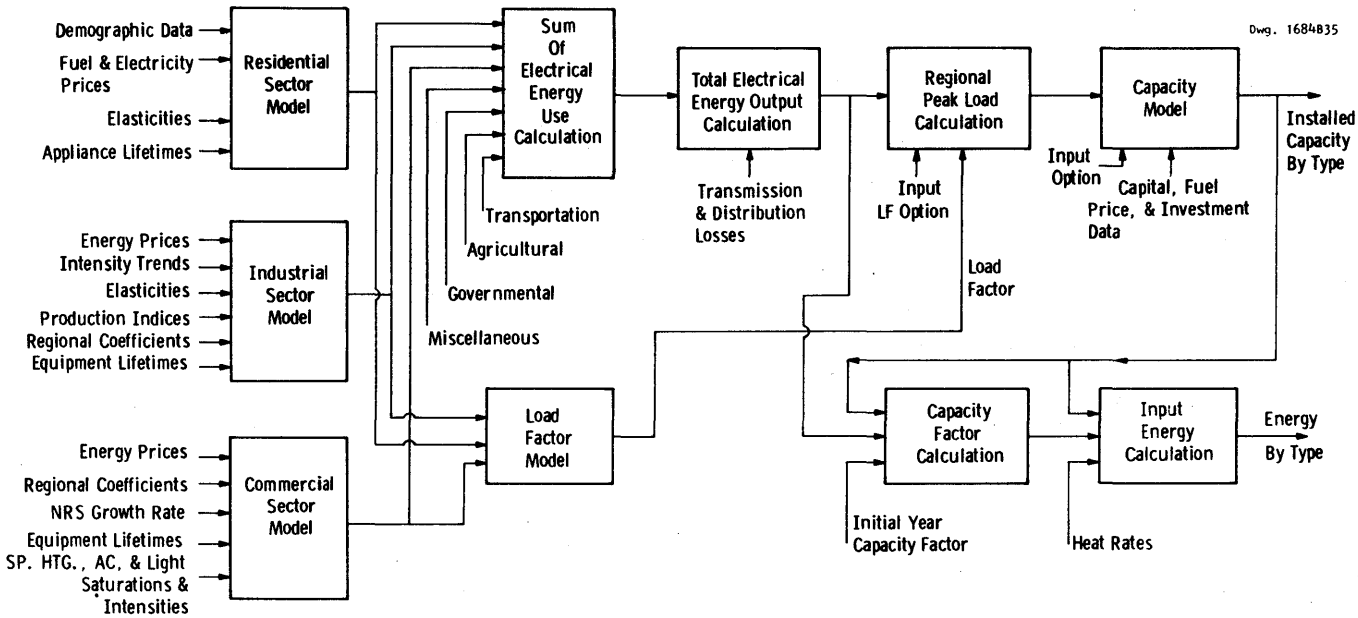


Figure 2—Power systems market forecast model

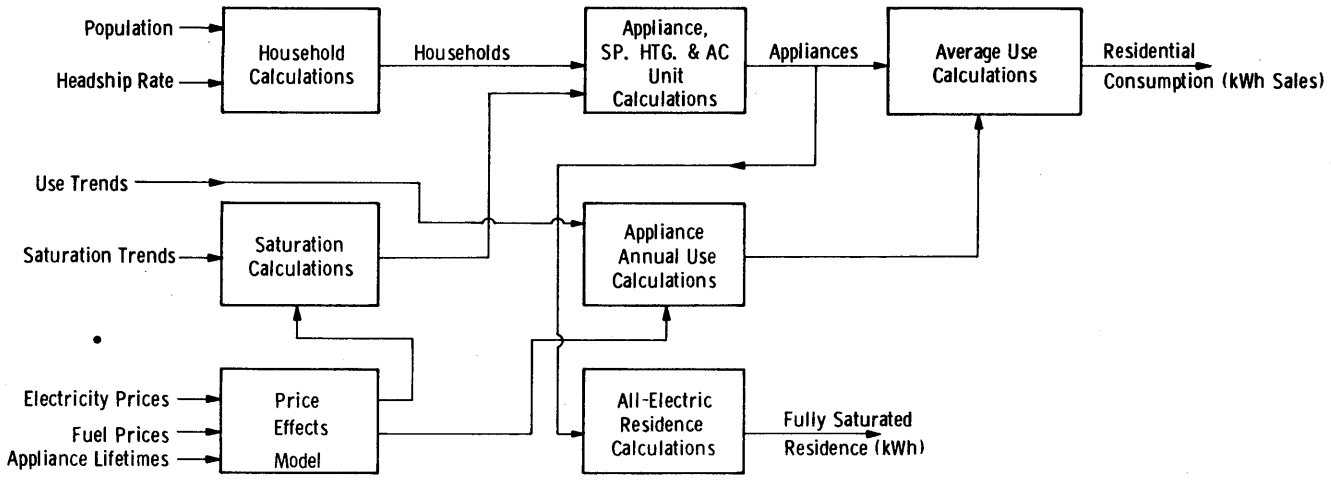


Figure 3—Residential sector model

where FRB_{NAT} is the Federal Reserve Board Indices, $VAREGSIC is the $ of value added by SIC by region, $VANATSIC is the $ of value added nationally by SIC, USESIC is the electricity use per unit output for each SIC, $IUSEA_t$ and $IUSEA_{t-1}$ are the industrial use price adjustments in years t and t−1, λ is the fraction of this price adjustment accomplished each year, RPEI and RPOI are the relative prices of electric and other fuels in the industrial sector to a base year, α_{UIE} and α_{UIO} are the price elasticities of use for electric and a composite for other fuels.

Commercial sector model

The commercial sector model, shown in Figure 5, uses national projections of non-residential structures or commercial buildings to estimate commercial floor space in use in any given year. Regional allocators developed from regional population and personal income forecasts[3] are used to estimate floor space in each NERC region. Input time trends are used to calculate both the lighting, space heating, air conditioning, and miscellaneous electricity use per square foot of floor space and the saturation of these uses. A price adjustment is made on total commercial demand. The equations used for the estimated KWH consumption are:

$$COMKWH=NRSGR*COMFLB*REGAL*\sum USEC*CSAT*COMPA_t \quad (6)$$

$$COMPA_t=(1-\lambda_c)COMPA_{t-1}+\lambda_c RPEC^{\alpha CE}*RPGC^{\alpha CG} \quad (7)$$

where NRSGR is national non-residential structures or com-

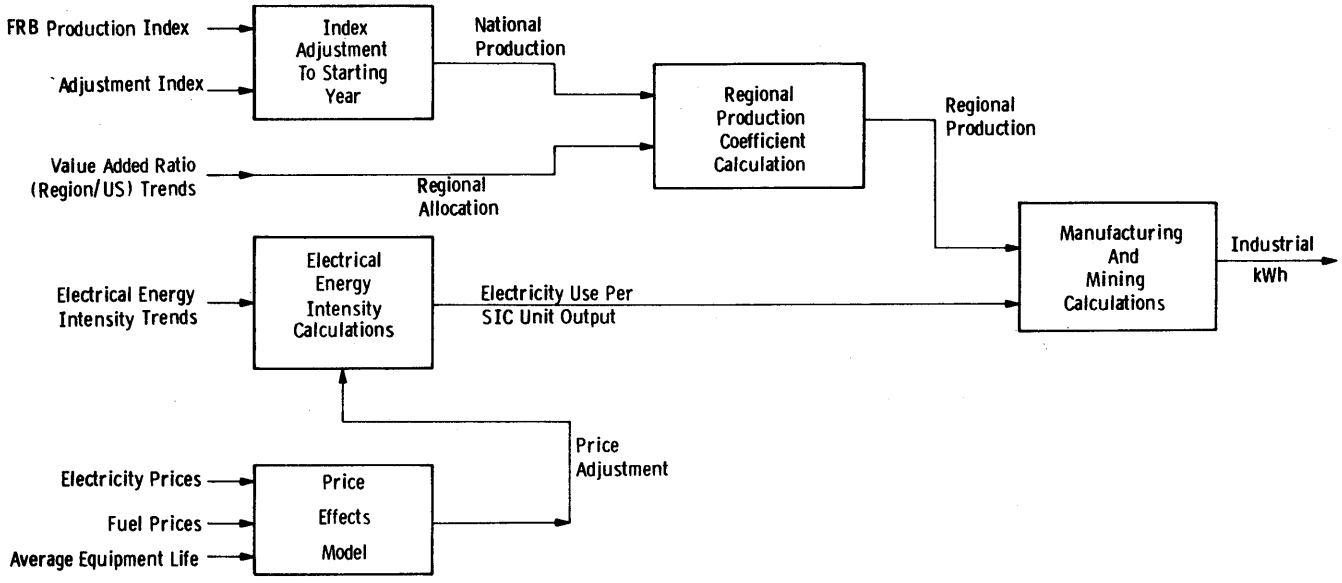


Figure 4—Industrial (mining & manufacturing) sector model

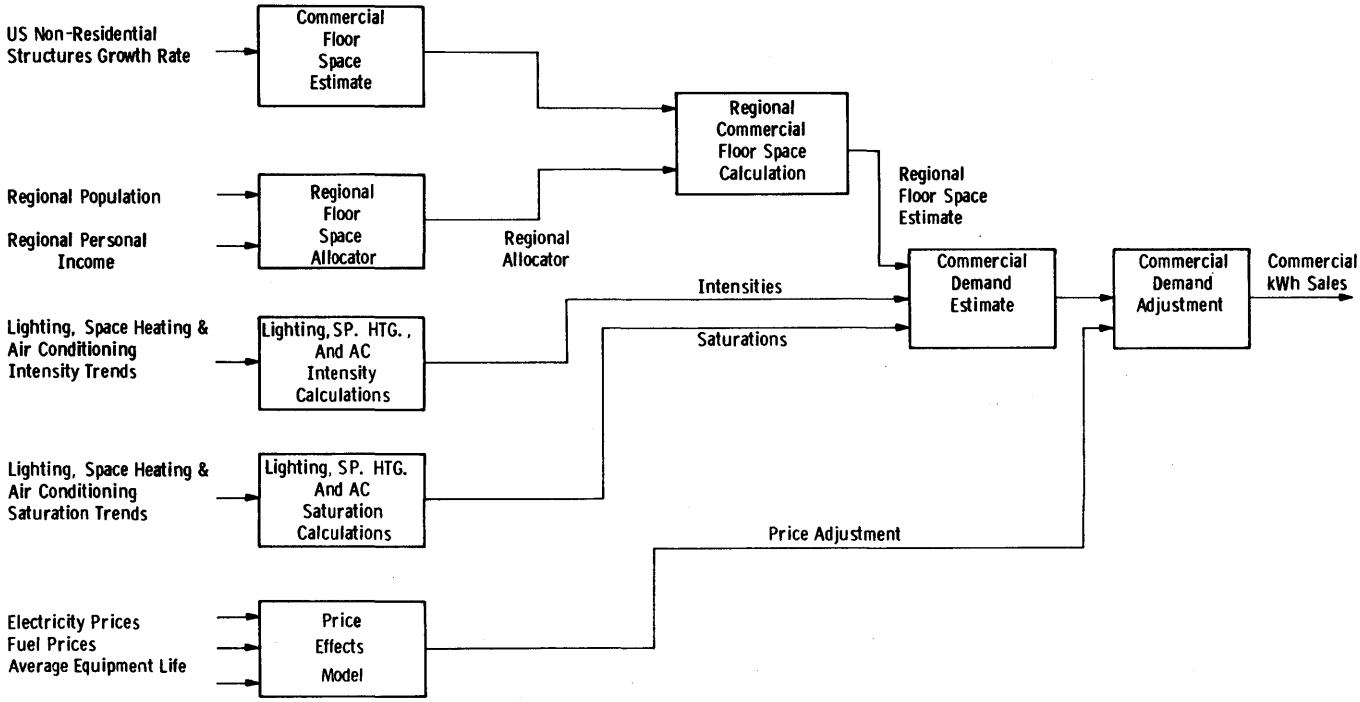


Figure 5—Commercial sector model

mercial buildings growth index relative to the base year. COMFLB is the commercial square feet of floor space in the base year, REGAL is the fraction of floor space allocated to each region, USEC is the KWH consumption per square foot of floor space for each specific use, $COMPA_t$ and $COMPA_{t-1}$ are the commercial price adjustments in year t and t−1, λ_c is the fraction of the price adjustment taking place each year, RPEC and RPGC are the relative prices of electricity and gas with respect to the base year, and α_{CE} and α_{CG} are the price elasticities of total use in the commercial sector.

Load factor model

Ideally, one would like to estimate and forecast the peak electric load independently from KWH sales. However, because of the scarcity of data with regard to individual loads and their coincidence on the system, this is impossible to accomplish in general. The approach used here is to relate peak load to residential, commercial, and industrial KWH sales, saturation of space heating and air conditioning, cooling and heating degree days. After obtaining peak load (PLW, PLS), the load factors (LFS, LFW) are calculated by relating to the total KWH output. The equations used are

$$PLW=\alpha_\omega RKWH+\beta_\omega COMKWH+\gamma_\omega INDKWH+\sigma_\omega RHS+\delta_\omega HDD \quad (8)$$

$$PLS=\alpha_s RKWH+\beta_s COMKWH+\gamma_s INDKWH+\sigma_s RAC+\delta_s CDD \quad (9)$$

$$LFS=(KWHOUT\div(8760\times PLS)) \quad (10)$$

$$LFW=(KWHOUT)\div(8760\times PLW)) \quad (11)$$

where PLW and PLS are winter and summer peak load, LFS and LFW are summer and winter load factor, KWHOUT is total KWH output, RSH is the saturation of residential electric space heating, RAC is the saturation of residential air conditioning, HDD and CDD are the regional average heating and cooling degree days.

The coefficients in equations (8) through (10) are estimated from historical data for each region. These independent variables were chosen because in the past, residential and commercial loads have contributed more to peak than industrial loads. Also, loads with larger weather-sensitive components have exhibited generally lower load factors.

Generation capacity model

Using total KWH output and regional load factor, peak load is computed for each region for each year of interest. With an estimate of peak load as input, the function of the generation capacity model is to determine how much capacity (if any) should be added in each region during each time period. If additions are required, then the model must calculate the type needed from a group of nine possible unit types: fossil (coal, oil, gas), nuclear, combustion turbine, combined cycle, hydro, pumped storage, and other.

The block diagram shown in Figure 6 demonstrates the functions performed by the generation capacity model. An initial estimate of the end-of-year capacity is calculated by

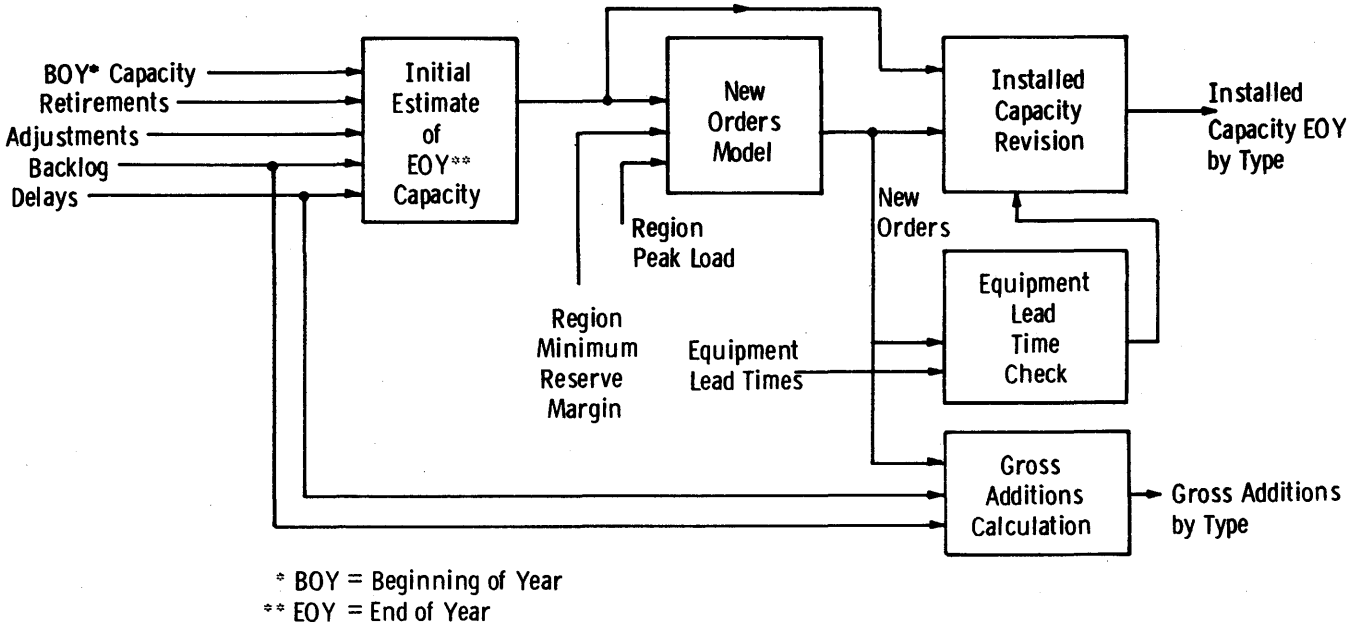


Figure 6—Generation capacity model

adding backlog, delays, and adjustments and subtracting retirements from the beginning-of-year capacity. Using this estimate of end-of-year (EOY) capacity with inputs of peak load and the minimum reserve margin in the region, a check is made to establish the sufficiency of the capacity. If the capacity is insufficient to meet peak load requirements with the minimum reserve margin, then new orders are generated for additional units. These new orders are then added to the initial estimate of the installed capacity to obtain the installed EOY capacity. These new order units are then checked against the equipment lead times to establish that sufficient time is available so that units being added can be delivered and installed. Gross additions for each type unit are calculated by adding backlog, delays, and new orders.

There are two options in the new orders model for calculating the type of units to be added. The first option simply provides for calculating the mix of new units using percentages estimated from the NERC reports. The second selects additional capacity from the nine types of units so that additions are matched to load requirements in a manner that minimizes cost.

BASE ELECTRICAL ENERGY AND PEAK LOAD FORECAST

To demonstrate the capabilities of the model, a base forecast has been prepared which uses the nine NERC data bases and aggregates the results from these regions to obtain a national forecast (Tables I, II and III). Because of space limitations, very little of the model output can be shown. Only a very brief summary of the base case is presented here.

The macroeconomic factors which drive the model are taken from the output of a macroeconomic model[4] so that consistency is established. For example, the industrial production, household growth, and commercial buildings growth are all consistent with a much larger set of economic conditions. The number of variables used directly as input to the electrical demand model from the macroeconomic model output is small (~25 time series). However, the macroeconomic model which derives these time series is comprehensive (over 1100 equations and identities), and, therefore, indirectly the influence of all pertinent economic factors are included.

National and regional KWH sales and output (sales+losses) are shown in Table I; historically for 1971-1976 and forecasted 1977-2000. Compounded annual growth in the period 1977-2000 is approximately 5 percent. This is considerably less than historic growth in the decades of the 60s and 70s where annual growth averaged 7 percent. The industrial sector is the major contributor to growth in this forecast based on the compounding effects of long-term expansion and increasing use of electricity per unit output in most industry groups. All NERC regions exhibit lower growth rates than historic and range from approximately 5.5 percent in the southwest to 3 percent in the northeast.

National and regional peak loads as forecasted by the model are shown in Table II. Actual values for the period 1971-1976 are shown in parentheses. Nationally, summer peak loads are forecasted to grow at about 4.6 percent annually and winter peak about 4.8 percent. Thus, peak is forecasted by this model to grow at a lower rate than KWH output. This can be explained by the major part of KWH output growth being industrial which contributes less to peak, and the application of load management techniques in controlling peak. Regionally, peak growth is highest in the

TABLE I.—National and Regional KWH Sales and Output

KILOWATT HOUR OUTPUT-BILLIONS

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	TOTAL
1971	167.600	138.200	278.000	318.300	129.800	121.700	94.100	61.500	307.900	1,617.100
1972	178.100	147.400	303.200	346.700	138.400	132.800	104.700	66.400	332.200	1,749.900
1973	189.000	158.500	325.400	384.400	146.400	140.700	110.400	69.900	344.300	1,869.000
1974	184.700	154.300	318.400	384.400	151.000	144.400	113.800	73.800	350.700	1,875.500
1975	183.800	154.400	319.500	396.300	155.200	156.600	115.100	77.100	362.200	1,920.200
1976	191.800	161.100	341.800	426.800	160.400	168.700	120.800	77.400	389.500	2,038.300
1977	202.800	171.700	368.300	458.400	173.200	180.800	129.900	82.500	420.000	2,187.600
1978	213.000	181.600	390.800	490.500	185.100	192.700	138.900	87.600	448.500	2,328.700
1979	222.600	191.100	412.100	521.500	196.500	204.500	148.000	92.400	476.000	2,464.700
1980	232.200	200.700	435.700	551.600	208.700	216.600	157.600	97.400	505.500	2,606.000
1985	278.900	250.500	547.000	721.200	269.200	282.700	211.100	122.800	661.200	3,344.600
1990	325.200	307.900	667.800	943.200	342.900	366.900	293.400	152.000	854.000	4,253.300
1995	372.000	371.800	801.600	1,221.900	428.200	470.400	384.900	184.900	1,084.000	5,319.700
2000	413.400	435.000	942.300	1,550.000	522.700	595.000	490.400	220.100	1,349.000	6,517.900

KILOWATT HOUR SALES-BILLIONS

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	TOTAL
1971	154.700	124.000	250.000	288.000	117.100	109.900	84.700	55.200	283.000	1,466.600
1972	161.700	132.100	272.000	313.700	124.200	119.800	92.700	59.200	302.400	1,577.800
1973	182.200	142.000	292.900	350.400	131.600	127.100	99.400	63.500	314.300	1,703.400
1974	167.300	139.600	288.600	351.000	136.000	129.400	101.900	66.600	320.300	1,700.700
1975	165.600	137.500	287.400	360.900	138.700	140.900	104.000	69.200	328.600	1,732.800
1976	172.300	146.500	311.000	388.400	147.300	151.700	110.500	69.400	345.900	1,843.000
1977	182.200	156.100	335.100	417.100	159.000	162.700	118.800	73.900	373.000	1,977.900
1978	191.400	165.100	355.600	446.300	170.000	173.600	127.100	78.500	398.300	2,105.900
1979	200.000	173.700	375.000	474.500	180.400	184.400	135.400	82.800	422.800	2,229.000
1980	208.600	182.400	396.500	501.900	191.700	195.500	144.200	87.300	448.900	2,357.000
1985	250.600	227.700	497.800	656.300	247.200	256.300	193.200	110.000	587.200	3,026.300
1990	293.000	280.400	607.600	858.200	314.900	332.100	255.600	136.200	758.500	3,836.500
1995	335.200	338.600	729.400	1,111.800	393.200	425.700	335.600	165.700	962.800	4,798.000
2000	372.500	396.100	857.400	1,410.400	480.000	534.400	428.900	197.300	1,198.100	5,875.100

TABLE II.—National and Regional Peak Load

SUMMER-PEAK LOAD (GW)

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	TOTAL
1971	29.315	25.991	45.571	57.973	26.410	24.654	18.839	12.034	53.365	294.152
1972	29.786	27.217	48.987	62.541	27.327	26.920	20.971	12.918	56.564	313.231
1973	34.595	30.152	53.409	68.915	29.417	28.700	22.857	13.891	58.755	340.693
1974	32.044	29.808	52.962	69.352	29.953	29.379	23.553	14.598	59.925	341.573
1975	32.255	30.128	53.666	72.196	30.643	31.990	24.223	15.158	62.178	352.438
1976	33.479	31.537	56.756	76.114	3 .488	34.611	25.998	15.125	65.979	371.087
1977	35.483	33.813	61.268	81.672	34.173	37.440	28.204	16.305	70.090	398.450
1978	37.283	35.565	64.662	86.499	36.072	40.261	30.426	17.476	74.303	422.547
1979	38.996	37.256	67.978	91.150	37.955	43.040	32.633	18.619	78.381	446.007
1980	40.724	38.980	71.543	95.609	39.870	45.943	34.966	19.761	82.589	469.986
1985	49.357	47.976	88.480	120.257	49.621	61.830	48.098	25.731	105.031	596.381
1990	57.911	57.986	105.951	150.644	60.816	81.816	64.993	32.618	131.659	744.397
1995	66.447	68.611	124.934	187.094	73.188	106.544	86.523	40.384	162.790	916.517
2000	74.452	79.126	144.712	228.774	86.632	135.378	111.857	48.719	197.899	1,107.549

WINTER-PEAK LOAD (GW)

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	TOTAL
1971	29.685	20.468	43.514	51.487	20.235	16.328	11.736	10.378	55.051	258.881
1972	30.969	22.501	48.356	57.386	22.290	18.533	13.958	12.036	58.240	284.270
1973	34.174	23.738	51.395	64.286	22.684	20.740	15.710	12.088	60.792	305.606
1974	32.488	24.308	51.623	65.613	24.384	21.502	16.737	13.234	61.453	311.342
1975	32.720	24.647	52.275	69.104	25.081	24.488	17.609	14.140	63.979	324.043
1976	34.227	26.918	56.764	74.563	26.826	26.685	19.064	14.001	67.444	346.491
1977	36.072	28.726	60.991	80.455	29.071	30.079	21.436	15.309	72.654	374.792
1978	38.138	30.798	64.884	86.323	31.358	33.252	23.605	16.477	77.450	402.286
1979	40.123	32.777	68.682	92.003	33.591	36.428	25.754	17.669	82.117	429.145
1980	42.112	34.783	72.749	97.545	35.914	39.627	27.990	18.825	86.922	456.467
1985	51.859	44.839	92.151	127.068	47.091	54.001	39.466	24.573	112.623	593.670
1990	61.428	56.079	112.259	163.235	59.874	70.397	53.446	30.721	142.975	750.416
1995	70.962	68.017	133.994	206.004	73.999	88.677	68.181	37.289	178.112	925.235
2000	79.949	79.725	156.692	254.639	89.292	106.150	85.577	44.066	217.732	1,113.821

TABLE III.—National and Regional Load Factors

LOAD FACTOR - SUMMER PEAK

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	NAT
1971	0.653	0.607	0.696	0.627	0.561	0.564	0.570	0.583	0.659	0.628
1972	0.683	0.618	0.707	0.633	0.578	0.563	0.570	0.587	0.670	0.638
1973	0.624	0.600	0.696	0.637	0.568	0.560	0.551	0.574	0.669	0.626
1974	0.658	0.591	0.686	0.633	0.575	0.561	0.552	0.577	0.668	0.627
1975	0.650	0.585	0.680	0.627	0.578	0.559	0.542	0.581	0.665	0.622
1976	0.654	0.583	0.687	0.640	0.582	0.556	0.530	0.584	0.674	0.627
1977	0.652	0.580	0.686	0.641	0.579	0.551	0.526	0.578	0.684	0.627
1978	0.652	0.583	0.690	0.647	0.586	0.546	0.521	0.572	0.689	0.629
1979	0.652	0.586	0.692	0.653	0.591	0.542	0.518	0.567	0.693	0.631
1980	0.651	0.588	0.695	0.659	0.598	0.538	0.515	0.563	0.699	0.633
1985	0.645	0.596	0.706	0.685	0.619	0.522	0.501	0.545	0.719	0.640
1990	0.641	0.606	0.720	0.715	0.644	0.512	0.515	0.532	0.740	0.652
1995	0.639	0.619	0.732	0.746	0.668	0.504	0.508	0.523	0.760	0.663
2000	0.634	0.628	0.743	0.773	0.689	0.502	0.500	0.516	0.778	0.672

LOAD FACTOR-WINTER PEAK

YEAR	NPCC	MAAC	ECAR	SERC	MAIN	SPP	ERCOT	MARCA	WSCC	NAT
1971	0.645	0.771	0.729	0.706	0.732	0.851	0.915	0.677	0.638	0.713
1972	0.657	0.748	0.716	0.690	0.709	0.818	0.856	0.630	0.651	0.703
1973	0.631	0.762	0.723	0.683	0.737	0.774	0.802	0.660	0.647	0.698
1974	0.649	0.725	0.704	0.669	0.707	0.767	0.776	0.637	0.651	0.688
1975	0.641	0.715	0.698	0.655	0.706	0.730	0.746	0.622	0.646	0.676
1976	0.640	0.683	0.687	0.653	0.683	0.722	0.723	0.631	0.659	0.672
1977	0.642	0.682	0.689	0.650	0.680	0.686	0.692	0.615	0.660	0.666
1978	0.638	0.673	0.688	0.649	0.674	0.662	0.672	0.607	0.661	0.661
1979	0.633	0.666	0.685	0.647	0.668	0.641	0.656	0.597	0.662	0.656
1980	0.629	0.659	0.684	0.646	0.663	0.624	0.643	0.591	0.664	0.652
1985	0.614	0.638	0.678	0.648	0.653	0.598	0.611	0.570	0.670	0.643
1990	0.604	0.627	0.679	0.660	0.654	0.595	0.627	0.565	0.682	0.647
1995	0.598	0.624	0.683	0.677	0.661	0.606	0.644	0.566	0.695	0.656
2000	0.590	0.623	0.686	0.695	0.668	0.640	0.654	0.570	0.707	0.668

Southwest (6.2 percent) and lowest in the mid-Atlantic (3.5 percent) and Northeast (3.5 percent).

National and regional load factors are shown in Table III. Nationally, summer load factor is forecasted to improve for the same reasons mentioned previously, that summer peak load is growing less than KWH output, industrial growth, and load management. National winter load factor is forecasted to decline because of the considerable increase in electric space heating, especially in the more moderate climates.

The low summer load factors forecasted by the model for ERCOT, SPP, and MARCA may not actually occur because, most probably, additional load management equipment and incentives will be applied to arrest these declining load factors.

REFERENCES

1. "CRA Studies Forecast Effectiveness of Electrical Demand Models," *CRA Research Review*, Charles River Associates, Inc., Cambridge, MA, Winter 1977.
2. "The Residential Demand for Energy: Estimates of Residential Stocks of Energy Using Capital," EPRI EA-235, January 1977.
3. "The Data Resources U. S. Long-Term Forecast," Data Resources, Inc., Winter 1977.
4. "State and Area Forecasting Service," Data Resources, Inc., May 1976.

Computer modeling of automotive engine combustion

by CHARLES K. WESTBROOK and LEONARD C. HASELMAN

Lawrence Livermore Laboratory
Livermore, California

INTRODUCTION

Using a general outline of the physical and chemical events which occur in an automobile combustion chamber during a complete engine cycle, we discuss some of the achievements and some of the limitations of computer modeling of automotive engine combustion. We point out that this type of modeling is often limited by lack of physical knowledge and by computer software and hardware restrictions. Still, within these constraints many types of combustion models have been able to make important contributions to present understanding of automobile engine combustion.

Although the internal combustion engine has been in use in automobiles for about a century, the current need for increased fuel economy and reduced chemical pollutant emissions creates many new problems for automotive engine design. In order to provide support for engine designers and to provide additional theoretical analysis and insight into the physical and chemical processes which are part of engine combustion, increased attention has recently been given to detailed computer modeling of automotive engine combustion.

Computer models of combustion processes are limited in three principal areas. The first of these areas involves inadequacies in fundamental descriptions of relevant physical systems. For example, fluid mechanical turbulence has a profound influence on many aspects of internal combustion engine performance, but the theory of turbulence and its effects on chemical reactions and flame propagation are rudimentary at present. In addition, theoretical descriptions of processes such as liquid atomization and other multiphase flow systems are generally inadequate. The second major limitation for numerical models of complex physical and chemical systems is that, even when the physical description is relatively complete, often there are serious deficiencies in the numerical methods available for solution of the relevant systems of equations. The comparatively recent development of "stiff equation" solution methods has made a significant improvement in the ability to handle the chemical kinetic rate equations of combustion chemistry, but other problems remain. In computations which include the effects of spatial transport of mass and energy, there are very large systems of coupled partial differential equations to be solved, often between 25 and 50 equations for each of as many as 10000 computational cells, for a total of nearly half a million differential equations. Matrix solution techniques for such systems are not available and most problems must be greatly simplified in order to find solutions. The final major limitation for many numerical models is the result of basic computer hardware constraints, including available core size, memory access times, and arithmetic operation speeds. These limitations restrict the spatial resolution, chemical kinetic detail, and geometrical generality of the problems which can be addressed with computer models. The combustion models which have been developed to date are all subject to each of these major limitations, and it is unlikely that this situation will change substantially in the near future. Still, within these limits, a great variety of useful and valuable information has been developed through numerical modeling of combustion problems.

In the internal combustion engine, fuel is introduced into the combustion chamber in any of a variety of forms, in the gas phase, as a liquid, or a combination of the two phases. The resulting mixture of fuel and air is then ignited, again in any of several methods, and the resulting flame or flames propagate through the combustion chamber, consuming fuel and oxygen and producing, as products of combustion, water, carbon dioxide, and an often bewildering variety of chemical pollutants. The energy released by the reactions is converted to mechanical energy as the piston moves downward. The flame can be extinguished by a number of mechanisms, by contact with the cold walls of the combustion chamber, by the rapid expansion of the combustion chamber as the piston moves downward, or by consumption of available fuel. Many forms of flame extinction or quenching leave unburned fuel in the combustion chamber which can eventually result in hydrocarbon pollutants in the atmosphere. The combustion products are then passed through an exhaust system where the product gases are often subjected to additional reaction environments, such as catalytic converters, in order to modify the exhaust composition. Seen in this general outline, it is readily apparent that no single numerical model can be expected to deal adequately with the physical and chemical details of the entire process. One approach to modeling the combustion process is that which is characterized as thermodynamic models.[1] In these models the gases are divided into two regions, burnt and unburnt; empirical relations are then used to determine the rate at

which gases go from unburnt to burnt. These models provide little information on the spatial and chemical details of the combustion process. Another approach is to break the broad total problem into a sequence of overlapping smaller problems which can be analyzed separately. Only by this type of fragmentation can any of the physical subsystems be made tractable for analysis. It is also common to use symmetry assumptions to simplify the geometrical aspects of many problems, since fully three-dimensional problems usually require a prohibitive amount of computer memory and CPU time.

The problem of fuel injection into a combustion chamber provides an instructive example of a number of these points. Numerical models of liquid fuel sprays[2-4] have assumed that the liquid spray consists of an array of independent spherical droplets. This type of treatment ignores the important process of the atomization of the initially continuous liquid jet, since there is no physical theory which properly describes liquid jet atomization. When the assumption is made that the jet does break up into individual fuel droplets, a purely computational problem arises. In one of the more ambitious droplet models to date,[2] the liquid droplet distribution function becomes a variable in eight independent dimensions, including three space coordinates, three velocity coordinates, the droplet radius, and the time. With even the minimum resolution in each dimension, this treatment requires the computational manipulation of a variable with a dimension of more than 1.5 million (decimal) words at each time

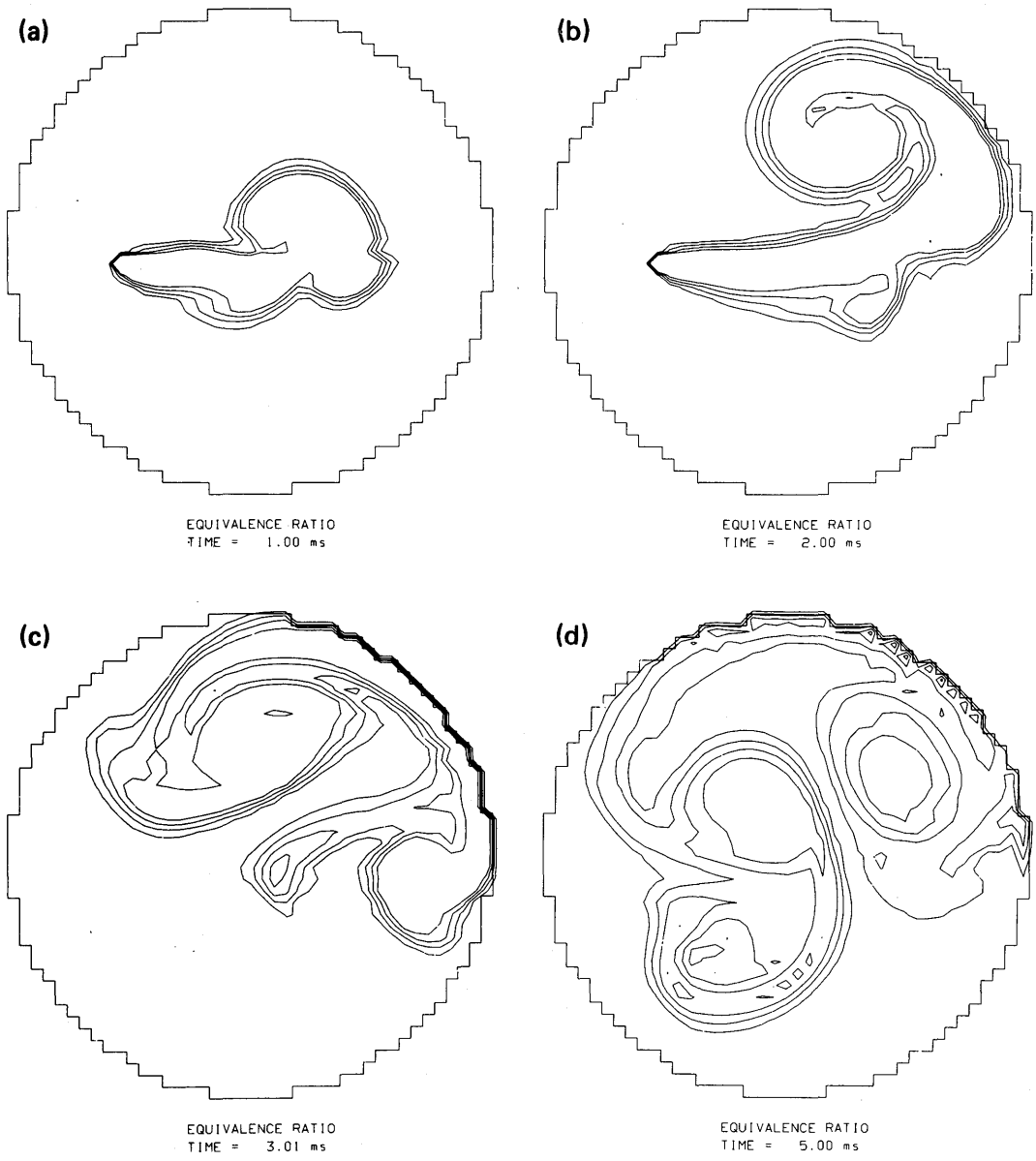


Figure 1—Contours of constant equivalence ratio for gas jet injection. Fuel-rich contours are near the center and fuel-lean contours are near the perimeter of the jet. Air swirl rate is 4000 RPM.

step. If assumptions are made with respect to physical symmetry, limiting the applicability of the model to somewhat idealized geometrical configurations, then the memory size requirements of the model are brought down to a more manageable 30,000-50,000 words, but some generality has been sacrificed.

Another injection model often used[3,5] consists of treating the injection as a gaseous jet into a gaseous medium. By eliminating the droplet phase a substantial reduction in the computational requirements is possible. In many practical applications, particularly in diesel engine chambers where conditions are close to the thermodynamic critical point of the fuel, the gas jet approximation is quite realistic. We can illustrate this approach with results computed by one of us (L. H.) for the case of a gaseous fuel jet injected into a swirling air flow. The combustion chamber is a thin circular disk representing a typical diesel combustion chamber near TDC, the point of maximum compression. Fuel is injected into the air flow which is swirling at a rate of 4000 RPM. The fuel is emerging from the injector at a velocity of 500 meters per second and is injected for 2 ms. As the gaseous fuel jet moves through the chamber, air and fuel gradually mix together. This mixing takes place preferentially at the edges of the gas jet where the shear force between the fuel and air is largest. In addition, the swirling air flow further enhances the fuel-air mixing. We can summarize the overall gas jet evolution by plotting contours of constant equivalence ratio ϕ in the combustion chamber at a sequence of times. The equivalence ratio is defined as the ratio of actual fuel concentration at a point to the fuel concentration which would exactly consume all of the oxygen at that same point. Therefore for $\phi>1$ the mixture is locally fuel rich, while for $\phi<1$ the mixture has excess oxygen. In Figure 1, we plot contours of constant equivalence ratio for values of $\phi=2.0$, 1.5, 1.0, and 0.5 at a sequence of times of 1 msec, 2 msec, 3 msec, and 5 msec after the beginning of injection. We see that the fuel jet travels across the chamber and gradually mixes with air. Large scale eddies are formed at each side of the jet due to air entrainment, and the swirling air flow deflects the jet in the direction of the swirl. These results have been compared and agree with experimental data.[3] The computer model has been used to examine the results of variations in a variety of quantities, including air swirl rate, injection velocity, and injector location. This type of parameter variation in a computer model can be much more economical than a corresponding experimental program and can be used to indicate promising conditions for a much smaller number of experiments. This process reduces the time and expense of a research program.

Once the fuel charge has been prepared, the mixture must then be ignited. The ignition process, either by compression as in the conventional diesel engine, or by spark discharge, presents many serious problems to a modeling effort. In the case of spark ignition, it is not yet clear how the electrical discharge initiates the combustion. It appears that, for a relatively short period of time, even the assumption of local thermodynamic equilibrium does not apply, i.e., that the translational, rotational, and vibrational temperatures of the gas are not all equal. There is an appreciable amount of ionization of the gas molecules, so the chemical kinetics properties of the gas are poorly understood. Also, fluid flow properties in the vicinity of a spark discharge are not always subsonic, and shock waves can be generated. In addition to these physical model inadequacies, ignition modeling presents serious numerical problems. The characteristic physical dimension of a spark region is very small, of the order of a millimeter or less. Yet the overall size of the combustion chamber is many times larger, indicating that lack of physical resolution may be a problem in computations. The ignition process is a difficult one for numerical modeling and a great deal of work is needed in this area.

Once the charge is ignited, a flame will propagate through the combustion chamber. The chemical kinetics of hydrocarbon fuel oxidation is another complex subject which is subject to both a lack of fundamental chemical information and by computational limitations. Detailed information on the reaction mechanism, intermediate chemical constituents, and elementary reaction rates is currently available for only the simplest of fuels. The chemical kinetics of methane (CH_4) involves as many as 25 chemical species and 50-100 elementary chemical reactions in a reasonably complete description of its oxidation and energy production.[6] Practical fuels such as gasoline involve many more constituents and reactions, many of which are unknown. As a result, simplifications are usually made in the expressions for the reaction mechanisms for almost all practical fuels.

Reaction mechanisms, exact or simplified, can then be coupled with fluid flow equations in a spatially varying, time dependent numerical model of flame propagation. Many such models exist. In one example, a comparison was made between combustion in a homogeneous charge engine and in a stratified charge engine.[7,8] Computed results indicate that charge stratification, combined with increased compression ratio, can make significant contributions towards reducing pollutant formation and improving fuel economy. Typical results are shown in Figure 2, indicating the reduction in carbon monoxide and nitric oxide levels as a result

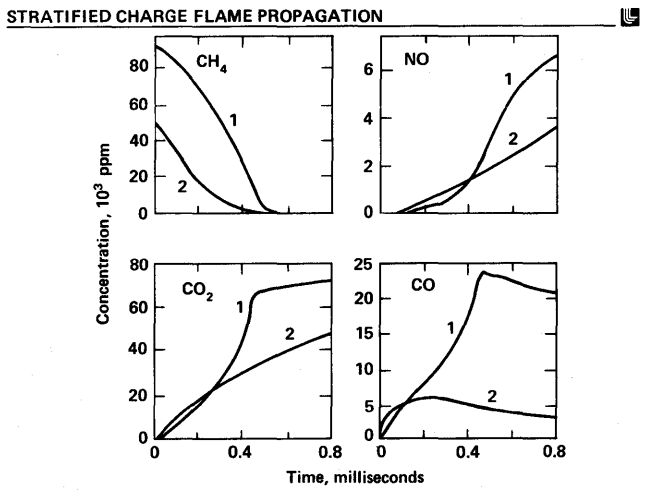


Figure 2—Total calculated concentrations of CH_4 (methane), NO, CO_2, and CO for homogeneous charge (1) and stratified charge (2) combustion.

of charge stratification. In addition, preliminary calculations[7,8] indicate that the stratified case may also provide increased fuel economy in terms of pressure increase per unit mass of fuel burned over the conventional homogeneous charge case.

In the analysis of stratified charge engine combustion, experimental techniques are not yet able to obtain enough data to adequately evaluate many performance characteristics. Time- and space-resolved information inside an engine chamber, including temperature, chemical species concentrations, and bulk fluid motion is not usually available, even with advanced laser diagnostic techniques. While this situation will eventually be remedied, at present computer modeling offers the only access to much of this type of information.

Another exceedingly important area of combustion research in automotive engines which is at present best approached by modeling studies, is that of flame quenching. Flame quenching can occur in several physically distinct regimes. Perhaps the best known quenching phenomenon results from a flame encountering a relatively cold wall of the combustion chamber. The gas temperature drops sharply in a boundary layer to the temperature of the wall; that temperature close to the wall is too low to sustain oxidation reactions, so the fuel which is in this region remains unburned. During the subsequent exhaust phase of the engine cycle these unburned fuel regions are swept out of the chamber and are a major source of the unburned hydrocarbon component of combustion pollutants. Modeling of this phenomenon is made difficult by two major factors. First, very fine physical resolution is needed in this type of problem, since the boundary layer has a thickness of less than a millimeter and this layer must be subdivided into a reasonable number of computational zones. Also, the mechanism of wall quenching almost certainly depends strongly on the details of the interaction between the thermal boundary layer and a number of elementary reaction rates, so this problem also requires a complete chemical kinetics model. Both the resolution and kinetics aspects of this problem require very large computers and sophisticated computational methods. Progress in modeling wall quenching processes is only now beginning to be made.

A physically distinct type of quenching can occur in a reciprocating engine, particularly under conditions of lean combustion and late ignition, where the piston motion rapidly increases the volume of the combustion chamber. This expansion lowers the reacting gas temperature and its density. Since the rates of the great majority of elementary chemical reactions decrease rapidly as the temperature and density are reduced, this volume expansion can significantly reduce the rate of fuel oxidation, even to the point of totally halting the flame propagation. Laboratory experiments and an accompanying numerical modeling study have together been able to provide a fairly complete physical description of this type of bulk quenching of hydrocarbon oxidation by rapid expansion of the reaction gases.[9]

Another type of numerical modeling of combustion processes in automotive engines is shown by an example of a multidimensional fluid mechanics model. It is currently not feasible to include a detailed chemical kinetics model with two or three independent space dimensions. This limitation can best be illustrated by remembering that in a typical kinetics model, approximately 20-30 differential equations must be solved, one for each chemical constituent. In a spatially varying model, these species equations must be solved for each computational cell, plus the usual fluid mechanics variables such as velocity and density. For a one-dimensional, time-dependent model with 100 cells this requires the solution of about 2500 coupled partial differential equations at each time step and takes up to an hour of CPU time on a CDC 7600 computer. For a computation with two space dimensions and 100 cells in each dimension, the same problem would consume over 100 hours, or four days of computer time, making such a computation practically unfeasible. As a result, almost all engine models which consider two space coordinates simplify the reaction model to one or two "global" reactions which are intended to give a reasonable representation of the chemical heat release which can be expected. Within this context, useful information about fluid flow properties which may not depend on the chemical kinetics details may be examined numerically. In an example of the use of this type of model, we compute the fluid mechanical motion of the gases in a motored engine chamber over a full four-stroke cycle. The engine is assumed to be axially symmetric, with a single intake-exhaust valve located on the chamber axis. The piston motion and valve timing represent an engine speed of 2500 RPM.

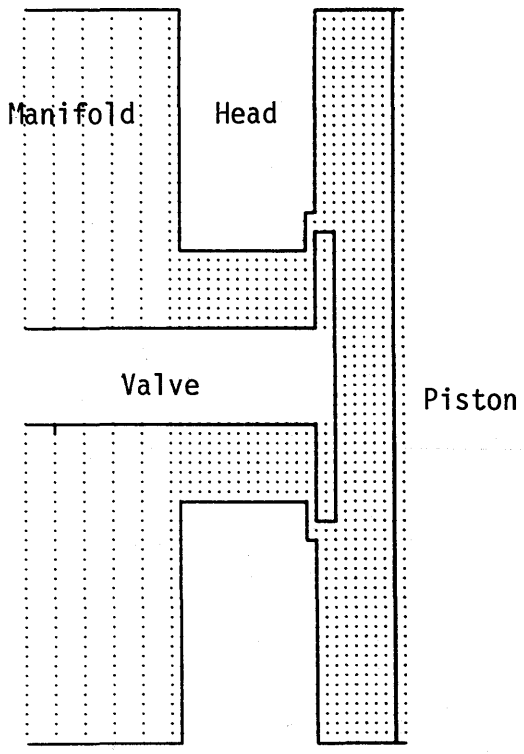


Figure 3—Initial grid for motored engine calculation. Only half of the grid is used for the calculation, it is reflected about the axis here for presentation purposes.

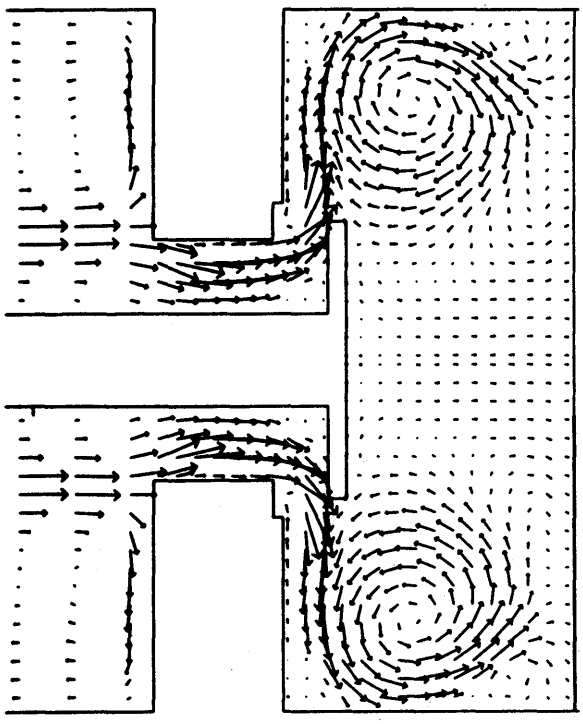


Figure 4—Time 4ms. Maximum velocity 174 m/s.

In Figure 3, we show the initial grid used in this calculation. The valve, head, piston, and cylinder walls are all represented as solid boundaries, with no heat conduction through them. The pressure at the left boundary is held constant at one bar, while the piston and valve move across the grid, which is stationary. This figure and the others in this series are taken from a sequence of computer-generated graphical output. This data has been put into the form of a continuous film which is a very useful and informative manner for presentation. Figures 4-15 are vector plots of the velocity for every other computational cell. For each plot the vectors are scaled linearly with the maximum velocity

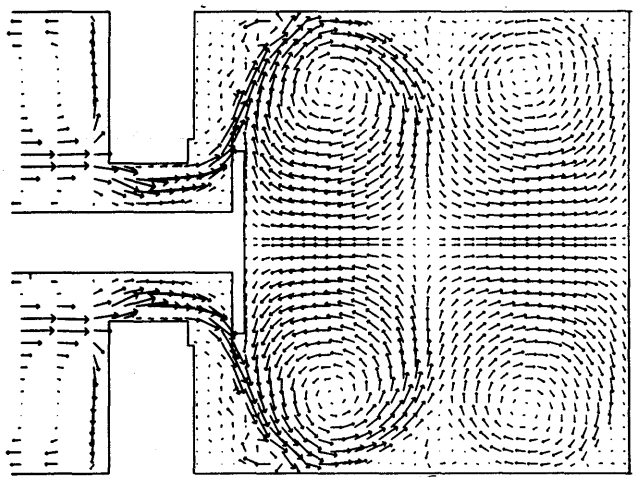


Figure 5—Time 8ms. Maximum velocity 178 m/s.

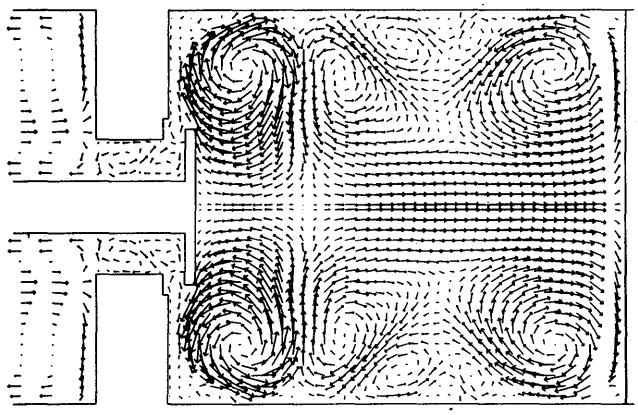


Figure 6—Time 12ms. Maximum velocity 70 m/s.

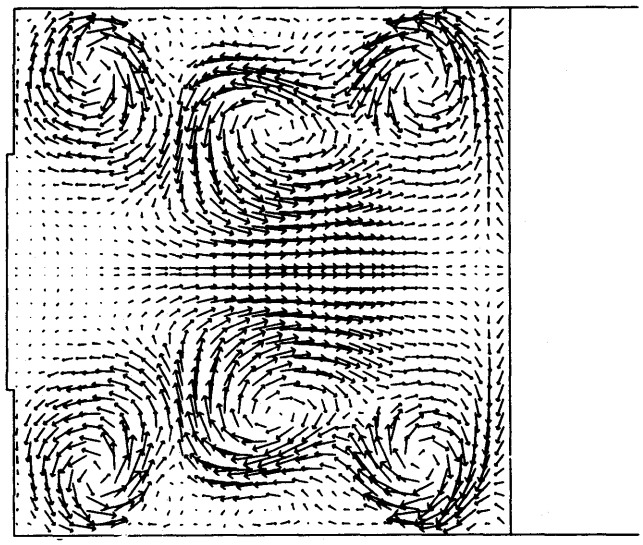


Figure 7—Time 16ms. Maximum Velocity 47 m/s.

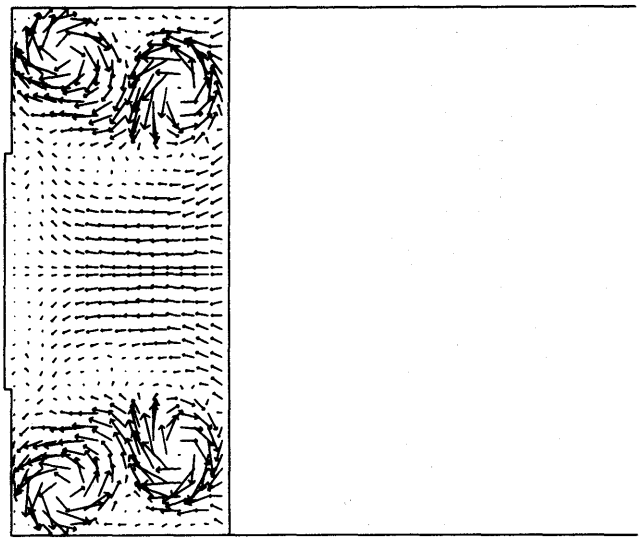


Figure 8—Time 20ms. Maximum velocity 42 m/s.

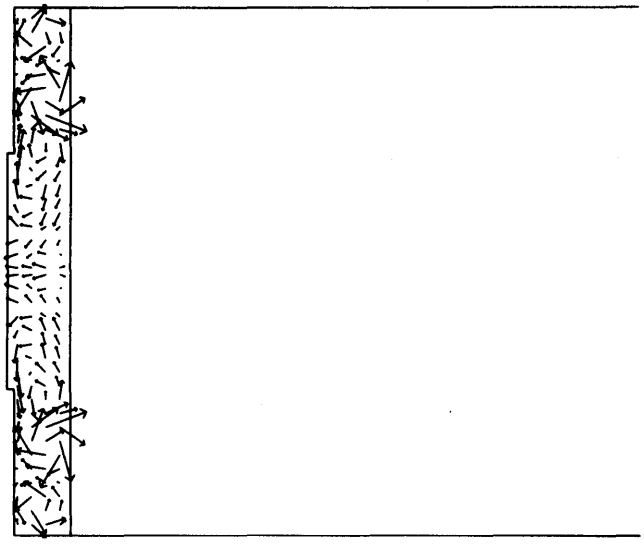


Figure 9—Time 24ms. Maximum velocity 14 m/s.

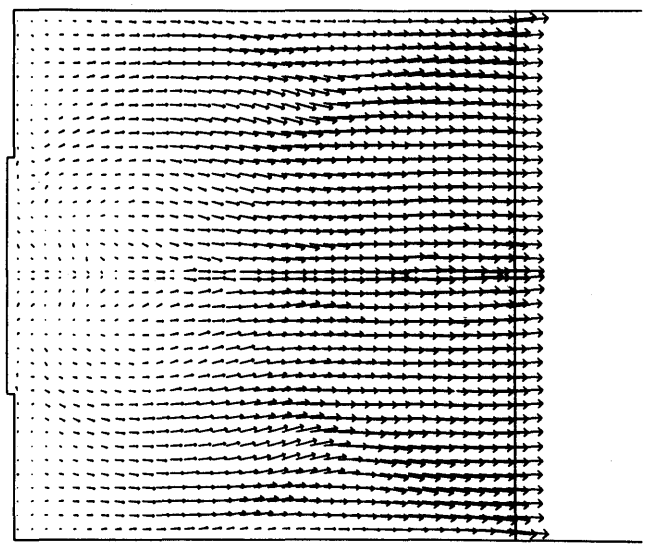


Figure 11—Time 32ms. Maximum velocity 10 m/s.

always being the same length. The plots are at 4 msec intervals, which is equivalent to 60° increments of crank angle.

The intake stroke is shown in Figures 4-6. At 4 msec a toroidal vortex is formed downstream from the valve and the flow velocity of the air through the valve has reached half the sound velocity of the air. This high velocity continues through 8 msec, by which time two toroidal vortexes have formed. At 12 msec, which is the bottom of the stroke, the flow through the valve has stopped and the peak velocity is approximately one fifth the sound speed. The flow in the cylinder chamber has now broken up into a number of small vortexes and is quite complex. At 14.4 msec the intake valve closes and it and the manifold are removed from the computational problem. At 16 msec the compression stroke has begun and the flow continues to decay since it no longer has the flow through the valve to drive it. Figure 7 and Figure 8 show the piston approaching the top and at the top of the compression stroke. At this time the grid resolution is fairly coarse and so the definition of the flow is rather poor. This could be corrected either by rezoning the grid or by using a grid which moves with the piston.

The expansion stroke is shown in Figures 9-11. On this stroke the gas motions caused by the intake valve have largely been dissipated, and subsequent gas motions are dominated by the motion of the piston. Through most of the expansion stroke the maximum velocity of the gas is close to the piston velocity.

At 33.6 msec the manifold and exhaust valve are put back into the grid. Figure 12 shows the calculation at 36 msec, at the bottom of the expansion stroke. The inflow of gas through the valve and the small vortex in the cylinder chamber are caused by the fact that the exhaust valve opens 2.4 msec before the end of the expansion stroke, so air can be drawn in from the manifold. Figures 13-15 show the exhaust stroke. The distinctive feature of this flow is that it is relatively simple. The flow reaches its maximum velocity at 44

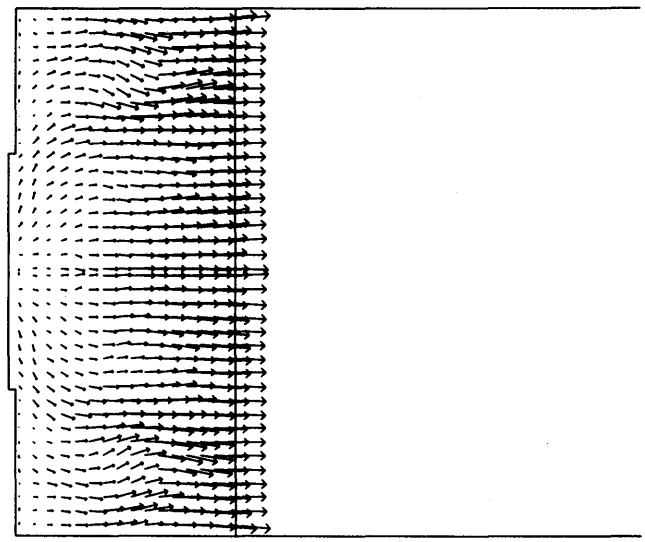


Figure 10—Time 28ms. Maximum velocity 12 m/s.

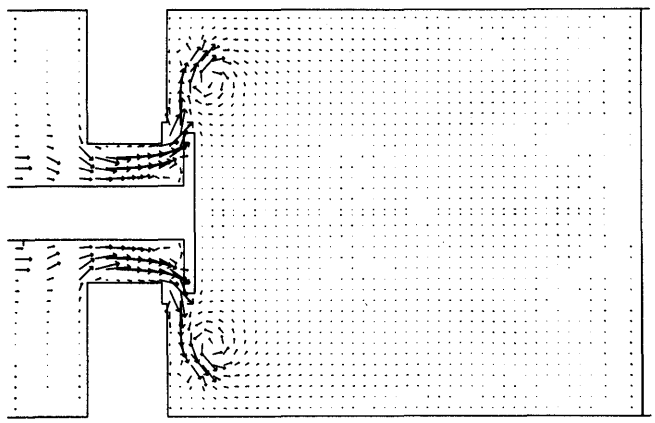


Figure 12—Time 36ms. Maximum velocity 48 m/s.

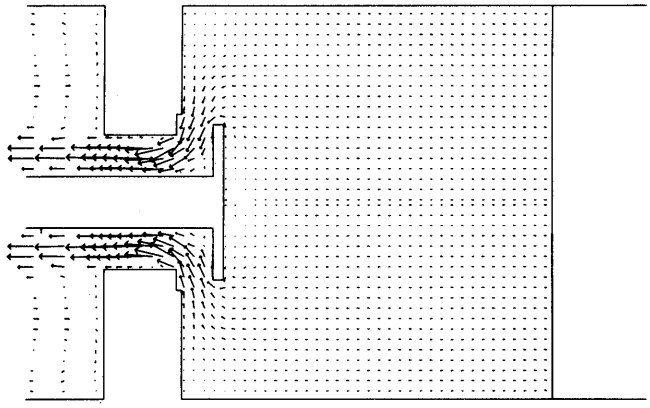


Figure 13—Time 40ms. Maximum velocity 113 m/s.

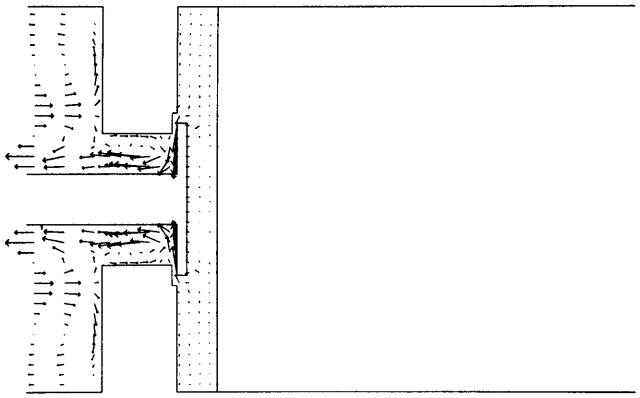


Figure 15—Time 48ms. Maximum velocity 46 m/s.

msec. The velocity at this time is two thirds the sound velocity.

These flow visualization calculations can consume significant amounts of computer time, particularly if a fine grid is needed to resolve the flow. Thus it is important in this kind of calculation to take maximum advantage of computer architecture and to have efficient algorithms for solving the equations. For example, the program used for the flow visualization calculation achieves a speed up of about a factor of two by using the parallel processing capabilities of the CDC-7600. Another factor of two was attained by using a more efficient algorithm for solving the equations. Thus a calculation which once took four hours of computer time can now be done in one hour.

In the examples presented of current research in computer modeling of automobile engine combustion, we have emphasized three major points. First, there are difficult theoretical problems which remain to be solved before many combustion problems can be effectively approached. For these problems, such as liquid atomization and turbulence, even the physical and mathematical formulations are not yet satisfactory. Major efforts are under way, directed towards solving some of these theoretical problems, and considerable progress can be anticipated in the next few years. Second, efficient numerical solution methods are needed in many cases where the mathematical model is satisfactory, but the computational methods are inadequate. Like in the case of chemical kinetics rate equations, where the development of stiff equation software made efficient solutions possible, problems remain which are primarily numerical in nature. One of the most prominent examples of this need is in the treatment of two-dimensional and three-dimensional geometrical problems involving spatial transport. At present, nearly all of the available solution algorithms for these problems are slow, unwieldy, and inaccurate, and a great deal of work needs to be done. The development of faster and larger computers, with new computer architectures, provides considerable promise in this area. Techniques which were not feasible on earlier computers may be attractive on vector pipeline or parallel processors. Finally, many additional limitations are placed on the productivity of computer combustion models by hardware limitations, particularly those of core size and computation speed. In this area as well, advanced computer development should be of great benefit to combustion research. The magnitude of the theoretical and computational tasks involved in modeling of automobile engine combustion makes it certain that this research will continue to require the most advanced computers and numerical techniques available. The automotive engine accounts for a significant part of current energy usage and an even larger share of the petroleum usage. Even small improvements in combustion properties can have a sizable impact in many areas, and computer modeling of automobile engine combustion has the capability to make significant contributions to improved combustion characteristics.

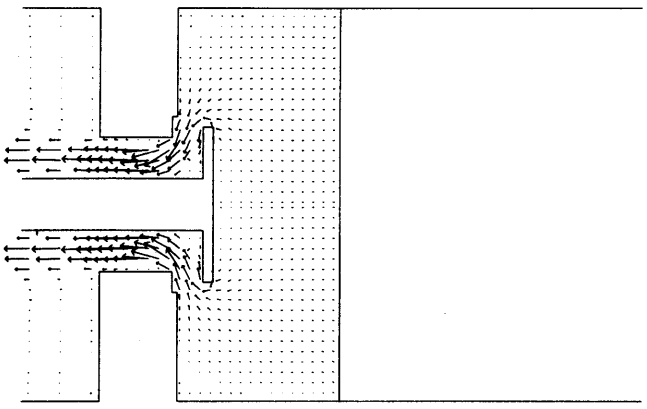


Figure 14—Time 44ms. Maximum velocity 201 m/s.

REFERENCES

1. Heywood, John B., "Modeling Combustion and Performance Characteristics of Internal Combustion Engines," *Proceedings of the 1976 Heat Transfer and Fluid Mechanics Institute,* 1976.
2. Westbrook, Charles K., "Three Dimensional Numerical Modeling of Liquid Fuel Sprays," *Sixteenth International Symposium on Combustion,* The Combustion Institute, Pittsburgh, Pa., 1977.
3. Haselman, Leonard C. and Charles K. Westbrook, "A Theoretical Model

for Two-Phase Fuel Injection in Stratified Charge Engines," Paper to be presented at the *Society of Automotive Engineers International Automotive Engineering Congress and Exposition,* February 1978.
4. Bracco, Frediano V., "Theoretical Analysis of Stratified, Two-Phase Wankel Engine Combustion," Combustion Science and Technology 8, 69, 1973.
5. Chiu, W. S., S. M. Shahed, and W. T. Lyn, "A Transient Spray Mixing Model for Diesel Combustion," *Society of Automotive Engineers Paper 760128,* February 1976.
6. Westbrook, C. K., J. Creighton, C. Lund and F. L. Dryer, "A Numerical Model of Chemical Kinetics of Combustion in a Turbulent Flow Reactor," to be published in *Journal of Physical Chemistry.*
7. Westbrook, Charles K., "Propagation of a Flame Through a Stratified Charge Combustion Chamber," University of California Lawrence Livermore Laboratory Report UCRL-79094, submitted for publication in *Acta Astronautica.*
8. Westbrook, Charles K. and Julius S. Chang, "A Theoretical Study of Flame Propagation Through Stratified Media," University of California Lawrence Livermore Laboratory Report UCRL-78925, presented at the Central States Section, The Combustion Institute, Cleveland, Ohio, March 1977.
9. Smith, O. I., C. K. Westbrook, and R. F. Sawyer, "Bulk Quenching of Hydrocarbon Oxidation in an Expanding Combustion Chamber," University of California Lawrence Livermore Laboratory Report UCRL-79580, presented at the 1977 Technical Meeting of the Canadian Section of the Combustion Institute, Banff, Alberta, May 26-27, 1977.

Computer modeling of automotive components and structures

by MOUNIR M. KAMAL and JOSEPH A. WOLF, JR.
General Motors Research Laboratories
Warren, Michigan

INTRODUCTION

After years of steady, predictable model changes, the American automobile industry is in the midst of the most intense product changeover in its history. By 1985, manufacturers must introduce new lines of vehicles which produce practically no pollutants, and which consume much less fuel. One obvious step in the solution is to resize the automobile, thereby producing a lighter, smaller vehicle, without sacrificing interior roominess and passenger safety. To accomplish the design of such a car, the structural engineer will need to use imaginative concepts and a variety of new materials. The computer is and has been an indispensable tool in carrying out these changes.

In the past, vehicle structural design was based on experience, extensive laboratory testing, and finally, proving ground evaluation and development. Analytic methods though available were extremely difficult, if not impossible to apply to the complex automobile structure (Figure 1). Emphasis, therefore, was on experimental determination of structural behavior and performance. However, the demands on the automobile designer increased and changed rapidly, first to meet new safety requirements and later to reduce weight in order to satisfy fuel economy requirements. Experience could not be extended to new vehicle sizes and performance data was not available on the new criteria. Mathematical modeling was therefore a logical avenue to explore. First, computer simulations of specific structural behavior were developed, especially in the area of automobile safety. Later the development of computer graphic packages allowed more accurate representation of automobile geometry with less engineering time required to generate the model. Most recently the finite element method, a computer dependent numerical technique, has opened up a whole new approach to vehicle structural design.[1]

The exponential increase in computerized analysis of automobile structures is reflected in the growing number of Society of Automobile Engineers (SAE) structural analysis papers presented in recent years. Over the last ten years, this increase has been at a compounded rate of 25 percent per year. A further sign of recent interest in finite elements and other structural analysis methods in the automotive industry can be found in the proceedings of two international conferences on Vehicle Structural Mechanics held in 1974 and 1977 by SAE.[2,3] The analytic methods discussed at these meetings made it possible to evaluate and optimize the automobile structural design.

Starting with the 1976 Cadillac Seville, all new General Motors automobiles have been evaluated and developed using the finite element method. Without these advanced computer aided analytic tools, it would not have been possible to make the progress that resulted in substantial weight reduction and still retain occupant safety, comfort and interior space requirements.

COMPUTER MODELS FOR VEHICLE SAFETY

In the early sixties, barrier impact testing was the only method available for evaluating the crashworthiness of a vehicle structure.[4] Later in that decade computer simulation techniques, based on test data, were used to help understand the dynamics of collision of both the vehicle[5] and the occupant.[6,7] In 1970 simulation was advanced to the point that the technique became a reliable design tool in the automotive industry. All these methods, however, required experimental inputs to represent the large plastic deformations in the structure. In particular, Kamal and Lin[5,8] represented the major vehicle structural components (such as the frame, sheet metal, drive line, fire wall etc.) as nonlinear resistances (Figure 2) whose force-displacement behavior are obtained from static crush tests. The body, engine, and transmission cross member are represented by lumped masses. The equations of motion are then solved in which the dynamic resistive forces of the crushing structures are each assumed to increase linearly with the instantaneous rate of deformation for that element of the structure. While these were important developments, there was still the need to solve the structural deformation problem analytically so that one could predictively assess the crashworthiness of a particular design. Given the irregular geometry of the automobile structure (Figure 1), the finite element method was a natural tool. Melosh and Kelly[9] outlined in 1967 the problems that would be encountered in applying the method to predict car crash

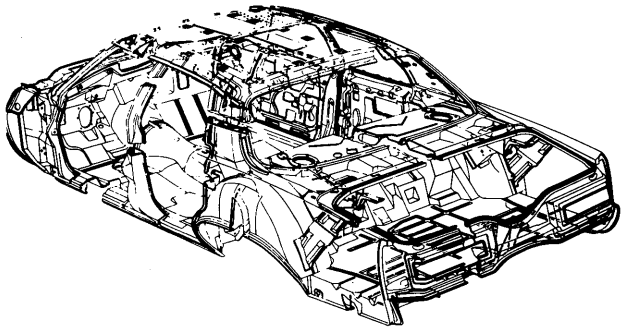


Figure 1—Typical automotive body structure

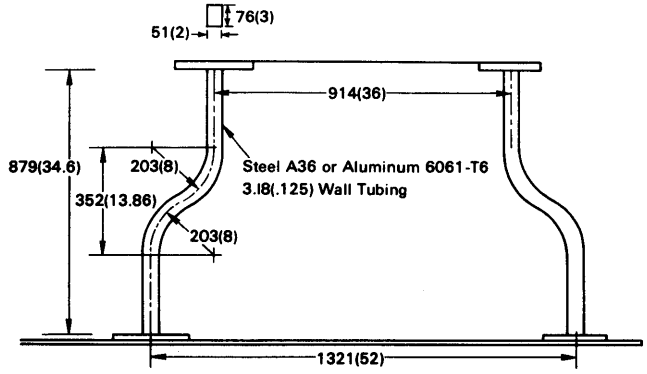


Figure 3—Dimension of curved box beam-column in mm (inches)

response. This, however, has yet to be accomplished for a full vehicle.

The structural problem to be solved is that of determining the dynamic forces and displacements for the automobile structure which is made up of shells, beams, corrugated panels, and irregular bars all subjected to large plastic deformations. Many of the components are made of thin-walled beams and columns whose sections collapse during these large high-rate deformations.

In the case of certain components or elements of an automobile structure, progress has been made. Ni[10] has modeled the impact response of a curved box beam-column using a finite element-finite difference technique. This type of structure (Figure 3) is used in the car frame and underbody structure. Good agreement with experimental measurements was obtained (Figure 4) for both aluminum and steel frames of various dimensions.

Chang[11] also discovered the importance of local deformation in his study of the automobile body structure. He presented an approximate technique for estimating the ultimate load-carrying capacity of the body structure without the need for experimental data (Figure 5). This procedure has been very useful in determining whether the body (or passenger compartment) will withstand the forces generated while the frontal structure crushes and decelerates the vehicle. It should be mentioned, however, that this is an ultimate load analysis quite different from what is needed in the frontal structure, namely the dynamic force-deformation function. Also metal panel deformations do not play as important a role in this problem as they do in frontal crush.

To summarize, computer simulation of vehicle impact is widely used to solve vehicle safety problems. Predictive analytic techniques are being developed, with progress being reported every year using numerical techniques heavily dependent on the digital computer.

VEHICLE MODELS FOR VIBRATION

Until the finite element method became available, dynamic modeling was on the basis of gross models with only a few degrees of freedom.[12] However, mathematical modeling using substructures or building blocks had been well understood for some time.[13] What was needed for implementation was the development of computers with large in-core storage capability to contain information on the large number of simultaneous equations or degrees of freedom required for a complete vehicle model. The final ingredient for success was the development of a "general purpose" structural computer code, which is user oriented, and has extensive documentation facilitating application by nonspe-

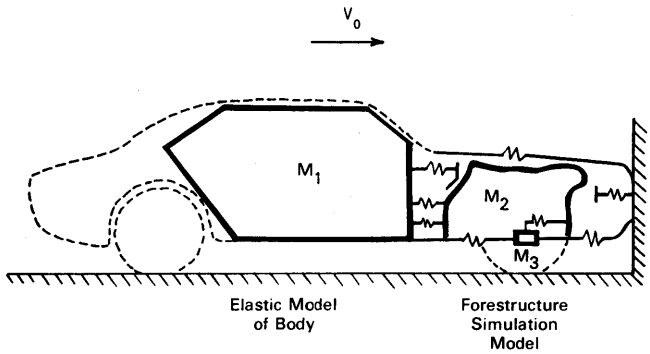


Figure 2—Mathematical model for the analysis of vehicle frontal impact

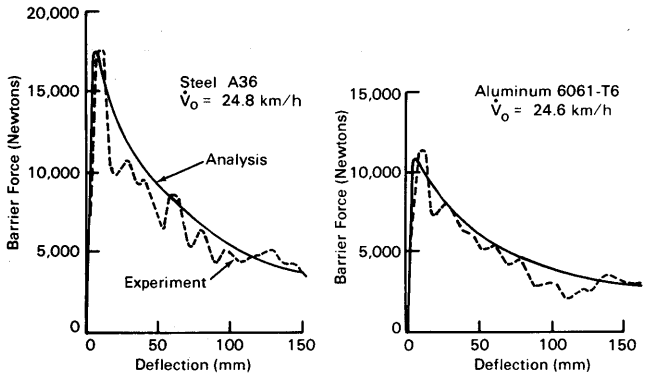


Figure 4—Comparisons of analysis and experiment for dynamic force-deflection relationship (steel, 15.4 mph and aluminum, 15.3 mph).

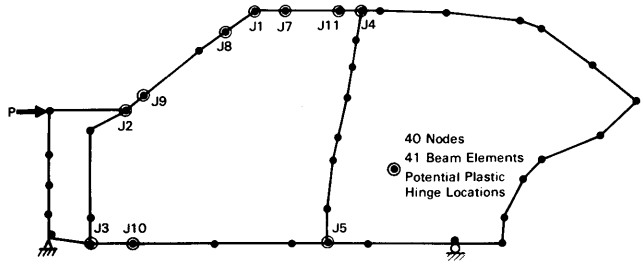


Figure 5—Finite element model of car body side structure

cialist design engineers. The NASA STRuctural ANalysis (NASTRAN) program is one such code.[14] A part of this user convenience has been the ease of model visualization and results interpretation arising from the choice of structural displacements as the unknowns in the formulation of the equations of motion.

Current methods

Within the last five years, complete vehicle structural models based on the finite element method have become a practical alternative to testing. More or less complete finite element models were described as early as 1970, by Selna and Salinas.[15] The method has been widely used both in the United States and overseas. This approach enables the analyst to generate a model based on structural blueprint dimensions and material properties, prior to any prototype build. Modeling is often done with enough detail so that the result may be used for examining local behavior as well as overall dynamic properties. The use of many thousand degrees of freedom (equations of motion) is not unusual. A typical model for a unit body car is shown in Figure 6. A typical beaming vibration mode shape resulting from computer finite element analysis of this car is shown in Figure 7.

The cost of performing a vibration analysis on a complete several thousand degree-of-freedom model can be prohibitively expensive even on a modern high-speed computer. Usually, dynamic degrees of freedom are needed only at points of significant mass and moment of inertia. Condensation techniques can be used in this case to reduce the unknowns to a more manageable number.[16]

Substructuring is also useful for cost and time savings.[13]

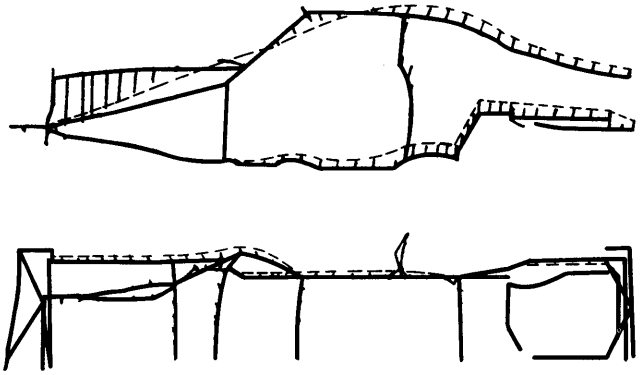


Figure 7—First beaming mode sprung mass model, 23.1 Hz

The key to success is the reduction in the total number of degrees of freedom (dof) as the model becomes more specific; that is, as we progress from a continuum (infinite dof), to a collection of finite elements (~2500 dof), to several substructures (~250 dof), to a structural system (~25 dof). This reduction procedure saves time and money in the computations without materially reducing our physical understanding. Substructuring is a good illustration of how one can extend the capabilities of the computer when the naive temptation is to represent more and more details in the hope of increasing the fidelity of the model.

Structural acoustics

Another area receiving considerable recent interest for modeling and computer solutions is low frequency vibrations (up to 200 Hz) of the interior volume of the passenger compartment. Finite element analysis is attractive for these systems of complicated geometry. Considering first the case of free vibrations, the passenger compartment air volume is subdivided into the usual finite element mesh, as shown in Figure 8. For low frequencies (less than 100 Hz), the sound pressure field has very little cross-body variation, and a two-dimensional model suffices.

The fluid mass and elasticity may be described relative to the finite element mesh as for a structural system. The calculations of natural frequencies and mode shapes then proceed in a straightforward manner.[17] The results are shown for four configurations of a station wagon in Figure

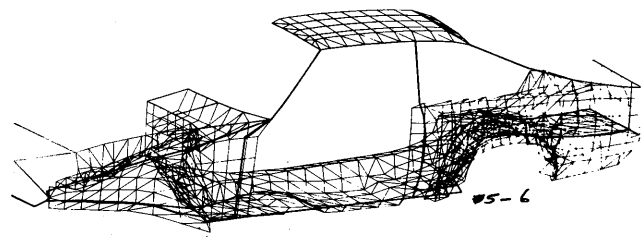


Figure 6—Model of unit body vehicle

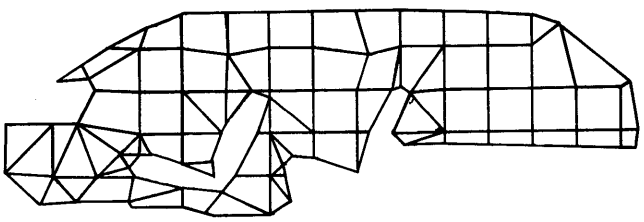


Figure 8—Finite element model of station wagon passenger compartment

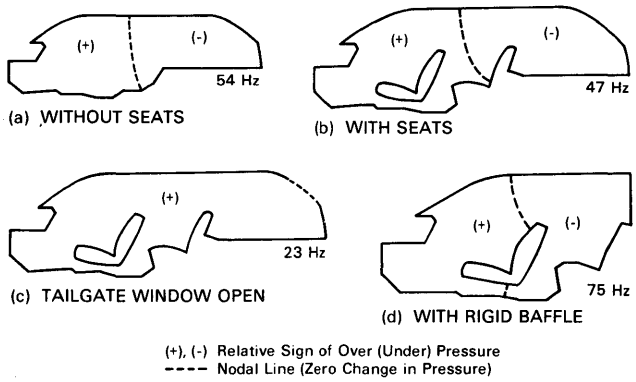


Figure 9—Acoustic mode shapes for four configurations of station wagon

9. The inclusion of flexible walls in the model is also easy to accomplish. Once the resonant properties of the compartment and surrounding structure are established, forced vibration analyses may be performed to predict the interior noise for various operating conditions.

Computer graphics

The magnitude of data input required for finite element models dictates the use of automated data generation and handling, usually using interactive computer graphics. At the General Motors Technical Center, the SMUG (Structural Modeling Using Graphics) Program[18] is based on earlier systems designed for recording and manipulating styling line data.[19] This program allows the analyst to generate a finite element model automatically from styling data during an interactive console session. The program output is a structural data file ready for input to NASTRAN. A typical plate model constructed using SMUG is shown in Figure 10.

Computer graphics is useful in several other areas, including checking finite element models for errors, and generating output plots of vehicle vibration modes. Graphics programs have also been developed for detecting interference among solids and surfaces, a prerequisite for allocating vehicle interior space efficiently.[20]

VEHICLE MODELS FOR RESIZING AND OPTIMIZATION

Resource and fuel savings are directly related to resizing and optimization of vehicle components. The availability of the computerized structural analyses discussed in previous sections of this paper led to the ability of automotive designers to resize production vehicles quickly without sacrificing interior roominess and passenger safety. For example, 1978 GM intermediate cars have been lightened by 310 kg (685 lb) on the average (Figure 11). The initial resizing resulted in a 68 kg (150 lb) savings. Subsequent compounding effects led to a total 15 percent mass reduction. These compounding effects result when weight reduction in one area permits reduction in another area. A good example is the bumper system. Clearly a narrower car needs a narrower bumper, saving 4 kg (9 lb) from resizing. Moreover, by reducing the total car mass through resizing of other components, the bumper requirements dropped, leading to an eventual overall savings of 30 kg (67 lb).

The next step in this process of design improvement might be to computerize the selection of a "best" design based on numerical criteria. This optimization process is an area of

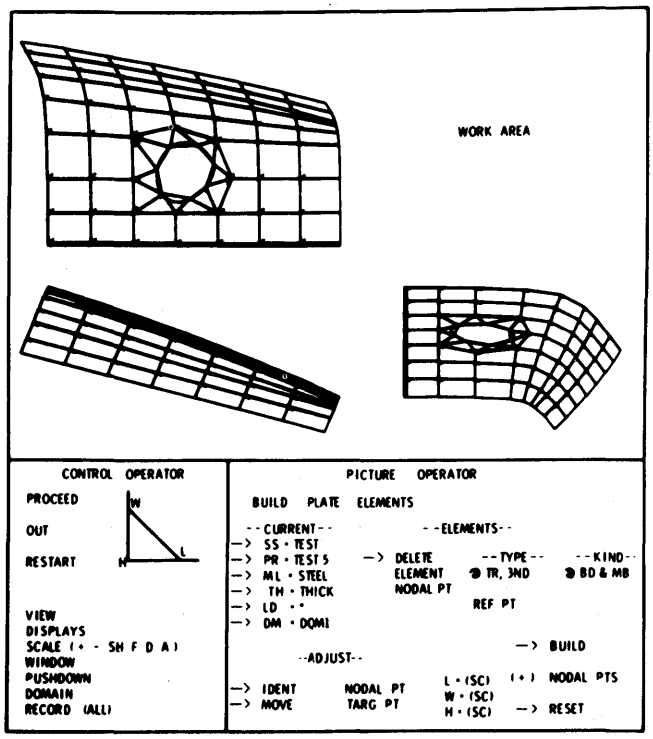


Figure 10—Display console screen image showing plate model

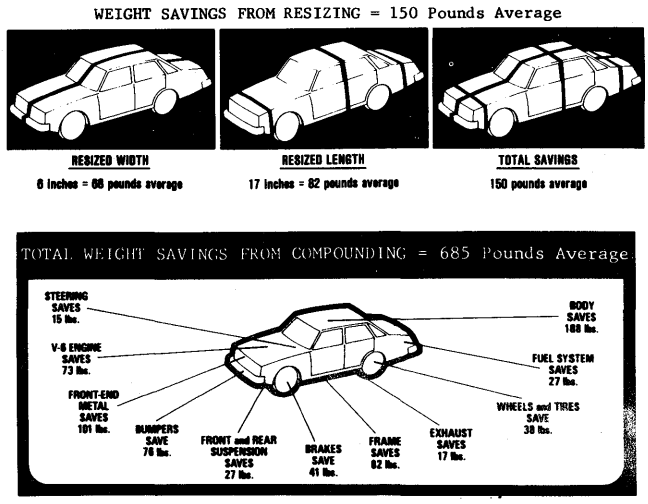


Figure 11—Weight savings from resizing

current research interest, and has resulted in recent papers on ride quality enhancement[21] and optimized vehicle crush resistance.[22] This concept of design through analysis has also been applied to the construction of an experimental automobile structure,[23,24] producing results of use in guiding production vehicle designers.

Material substitution studies

Another aspect of car design which can produce large mass and energy savings is that of material substitution. Using the technology of computer structural analysis and given explicit design criteria, the designer can quantify his engineering judgment regarding the benefit of a material substitution. For example, the effect on bending and torsion vibrations produced by a change in the mass of a "non-structural" hood panel of aluminum, or a change in the stiffness of a frame rail can be determined quickly using the computer. The various types of structural requirements to be considered in a material substitution study are discussed in a recent paper by Chang and Justusson.[25]

Computer stress analysis

In addition to the overall stiffness and dynamics computer models which have proved essential for vehicle resizing and material substitution studies, there has been considerable usage of finer mesh analysis models for computer stress analysis studies. Several papers representative of industry accomplishments in providing lighter, more efficient component designs appear in References 2 and 3. Problems dealt with include stress analysis of doors and other sheet metal parts, thermal stresses in brake drums, engine and transmission component design, stress analysis of a rear axle carrier, stress and deflection analyses of truck and tractor roll-over protection structures, calculation of gear stresses, stress and deflection analyses of (possibly nonlinear) suspension springs, and turbine and radial compressor component analysis.

FRICTION AND DRAG REDUCTION

Friction and drag reduction is another area of opportunity for the automotive engineer responsible for producing an energy and resource efficient vehicle. Because of the high degree of sophistication needed to model these phenomena, this area is one where the computer has already been of considerable use and has great future potential. This section will be divided into three parts, discussing aerodynamic drag, tires and lubrication.

Aerodynamic drag

Because of its timeliness and importance in fuel economy studies, this subject was the topic of a recent GM Research Laboratories symposium.[26] For current cars, the rolling resistance (due to the mechanical parts of the vehicle) and aerodynamic drag are about equal near 50 km/h (30 mph), but the aerodynamic losses are twice the mechanical losses at 90 km/h (55 mph).[27] As mechanical improvements occur, this trade-off speed will drop, emphasizing the aerodynamic losses more in the future.

The potential for energy savings is substantial. A 40 percent reduction in aerodynamic drag is felt to be feasible by aerodynamicists,[27] and might produce a 15 percent increase in fuel economy. Although the opportunities for energy savings are great in this area, direct computer assistance has been small to date due to the turbulent and usually unsteady nature of the three-dimensional flow field. Problems and prospects for analytical solutions in this area are discussed by Landahl[28] and by Hirt and Ramshaw.[29] At present, a finite difference technique has been applied to some problems, with finite element methods offering a second approach in the future.

Tires

Tire mechanics has been an active research area recently, and was the subject of a short course at the University of Akron in 1977. The variety of available numerical simulation techniques is discussed in one section of the course notes.[30] More specifically, because tire rolling losses are an important energy dissipator at all speeds, the relationship between tire rolling losses and fuel economy was the subject of a recent SAE-DOT research and development planning workshop. Discussion included the topics of numerical simulation of rolling tires and an analytical method for tire power loss calculations.[31]

Lubrication

The refinement of modern lubrication theory is of importance to the automotive engineer in reducing internal friction losses to the theoretical minimum. These losses occur not only in bearings, but also as friction between piston rings and cylinder wall (=25-50 percent of engine mechanical losses). Progress in this area has been impressive, in good measure due to the availability of large computers. Research on the film profiles which minimize the coefficient of friction and the total friction force for a given load in a slider bearing[32] are being extended to consider minimum friction loss in two-dimensional slider bearings, and in non-steady lubrication (i.e., piston rings sliding in an engine cylinder[33]).

Analysis of journal, thrust, and slider bearings has now been brought to the point of permitting a design engineer to correctly size such components interactively using computer aided design programs. For given geometry and material inputs, the computer immediately displays operating load, temperature, and power loss results. The designer can quickly refine the design to fit his needs.[34]

SUMMARY

Computer application in the automotive industry has had a rapid growth in the past 15 years. The world energy problem is but one of the challenges of recent times following safety and emissions. Computer models and methods are being used today to reduce weight, resize the automobile, reduce friction of moving parts, reduce aerodynamic drag, substitute materials, and assure product reliability and safety. This trend will continue and intensify in automotive engineering as progress is made in both computer hardware and the applied sciences.

REFERENCES*

1. Kamal, M. M. and J. A. Wolf, Jr., eds., "Finite Element Models for Automotive Vehicle Vibrations," *Finite Element Applications in Vibration Problems,* ASME, New York, 1977, pp. 67-92.
2. *Proceedings of the International Conference on Vehicle Structural Mechanics,* P-52, SAE, Warrendale, PA, 1974.
3. *Proceedings of the Second International Conference on Vehicle Structural Mechanics,* P-71, SAE, Warrendale, PA, 1977.
4. Wilson, R. A., "A Review of Vehicle Impact Testing: How It Began and What is Being Done," Paper No. 700403, Int'l Automobile Safety Conf. 1970, Belgium, also *Transactions of the SAE,* Vol. 79, 1970, pp. 1453-1467.
5. Kamal, M. M., "Analysis and Simulation of Vehicle to Barrier Impact," Paper No. 700414, Int'l Automobile Safety Conf., 1970, Belgium, also *Transactions of the SAE,* Vol. 79, 1970, pp. 1498-1503.
6. Martin, D. E. and C. K. Kroell, "Vehicle Crush and Occupant Behavior," Paper No. 670034, *Transactions of the SAE,* Vol. 76, 1967, pp. 236-258.
7. Egli, A., "Stopping the Occupant of a Crashing Vehicle—A Fundamental Study," Paper No. 670038, *Transactions of the SAE,* Vol. 76, 1967, pp. 259-289.
8. Lin, K. H., "A Rear-End Barrier Impact Simulation Model for Uni-Body Passenger Cars," Paper No. 730156, *Transactions of the SAE,* Vol. 82, 1973, pp. 628-634.
9. Melosh, R. J. and D. M. Kelly, "The Potential for Predicting Flexible Car Crash Response," Paper No. 670895, SAE Automotive Safety Dynamic Model Symposium, P-21, 1967, also *Transactions of the SAE,* Vol. 76, 1967, pp. 2835-2842.
10. Ni, C. M., "Impact Response of Curved Box Beam-Columns With Large Global and Local Deformations," AIAA Paper No. 73-401, *Proceedings of the 14th Structures, Structural Dynamics, and Materials Conference,* Williamsburg, VA, March 1973.
11. Chang, D. C., "A Design-Analysis Method for the Frontal-Crush Strength of Body Structures," SAE Paper No. 770593. In Reference 3, pp. 33-41.
12. Vail, C. F., "Dynamic Modeling of Automobile Structures from Test Data," *System Identification of Vibrating Structures,* W. D. Pilkey and R. Cohen eds., ASME, New York, 1972, pp. 149-177.

* In this survey paper, no attempt has been made to give complete technical documentation. We have relied primarily on the published work of our General Motors Research Laboratories colleagues to provide illustrative references. The interested reader is directed to extensive reference lists in References 1-3, 12, and 26.

13. Hurty, W. C., "Introduction to Modal Synthesis Techniques," *Synthesis of Vibrating Systems,* V. H. Neubert and J. P. Raney eds., ASME, New York, 1971, pp. 1-13.
14. McCormick, C. W., *The NASTRAN Users Manual,* SP-222(01), NASA, Washington, D. C., 1973.
15. Selna, L. and D. Salinas, "Dynamic Analysis of Automotive Structural Systems," Paper No. 700944, *Transactions of the SAE,* Vol. 79, 1970, pp. 2521-2528.
16. Nelson, M. F., "The Use of Condensation Techniques for Solving Dynamics Problems," SAE Paper No. 740330, in References 2, pp. 145-153, also *Transactions of the SAE,* Vol. 83, 1974, pp. 1435-1444.
17. Wolf, J. A., Jr., D. J. Nefske, and L. J. Howell, "Structural-Acoustic Finite Element Analysis of the Automobile Passenger Compartment," Paper No. 760184, *Transactions of the SAE,* Vol. 85, 1976, pp. 857-864.
18. Leverenz, R. K., et al., "Using Interactive Graphics for the Preparation and Management of Finite Element Data," Paper No. 740344, in Reference 2, pp. 279-285, also *Transactions of the SAE,* Vol. 83, 1974, pp. 1542-1548.
19. Devere, G. S., B. Hargraves, and D. M. Walker, "The DAC-1 System," *Datamation,* Vol. 12, No. 6, 1966, pp. 37-47.
20. Boyse, J. W., "Interference Detection Among Solids and Surfaces," Research Publication GMR-2740, General Motors Research Laboratories, Warren, Michigan, July 1977.
21. Baum, J. H., J. A. Bennett, and T. G. Carne, "Truck Ride Improvement Using Analytical and Optimization Methods," SAE Paper No. 770609, in Reference 3, pp. 211-220.
22. Bennett, J. A., K. H. Lin, and M. F. Nelson, "The Application of Optimization Techniques to Problems of Automotive Crashworthiness," SAE Paper No. 770608, in Reference 3, pp. 203-210.
23. Skattum, K. S., J. F. Harris, and L. J. Howell, "Preliminary Vehicle Structural Design for Comparison with Quantitative Criteria," Paper No. 750136, in *Vehicle Analytical Design for Structural Criteria—An Overview,* SP-392, SAE, pp. 31-42, 1975, Warrendale, PA, also *Transactions of the SAE,* Vol. 84, 1975, pp. 650-661.
24. Augustitus, J. A., M. M. Kamal, and L. J. Howell, "Design Through Analysis of an Experimental Automobile Structure," SAE Paper No. 770597, in Reference 3, pp. 69-81.
25. Chang, D. C. and J. W. Justusson, "Structural Requirements in Material Substitution for Car-Weight Reduction," Paper No. 760023, *Transactions of the SAE,* Vol. 85, 1976, pp. 66-78.
26. Sovran, G., T. A. Morel, and W. T. Mason, Jr., eds., *Aerodynamic Drag Mechanisms of Bluff Bodies and Road Vehicles,* Plenum Press, New York and London, 1978.
27. Hucho, W.-H., "The Aerodynamic Drag of Cars: Current Understanding, Unresolved Problems, and Future Prospects," in Reference 26, pp. 7-44.
28. Landahl, M. T., "Numerical Modeling of Blunt-Body Flows—Problems and Prospects," in Reference 26, pp. 289-311.
29. Hirt, C. W. and J. D. Ramshaw, "Prospects for Numerical Simulation of Bluff-Body Aerodynamics," in Reference 26, pp. 313-355.
30. Padovan, J., "Advanced Tire Models," in *Tire Mechanics,* Department of Special Programs and Department of Mechanical Engineering, University of Akron, 1977.
31. *Proceedings of the Symposium on Tire Rolling Losses and Fuel Economy,* SAE-DOT, Cambridge, Mass., to be published.
32. Rohde, S. M., "The Optimum Slider Bearing in Terms of Friction," *Transactions of the ASME,* Vol. 94, Series F, Journal of Lubrication Technology, 1972, pp. 275-279.
33. Rohde, S. M. and G. T. McAllister, "Some Preliminary Results on Dynamically Loaded Minimum Power Loss Lubricated Contacts," *Letters in Applied and Engineering Sciences,* Vol. 5, 1977, pp. 409-415.
34. Rohde, S. M. and K. P. Oh, "Some Current Trends in the Interactive Computer-Aided Design of Fluid Film Bearings," *Computer-Aided Design of Bearings and Seals,* F. E. Kennedy and H. S. Cheng, eds., ASME, New York, 1976, pp. 43-65.

The development and application of a mathematical model of enhanced oil recovery

by HARVEY S. PRICE

INTERCOMP Resource Development and Engineering, Inc.
Houston, Texas

INTRODUCTION

A great deal of attention has recently been focused upon "enhanced oil recovery"; that is, the application of recently developed processes for recovering oil that would normally remain in a reservoir following depletion by conventional techniques. Since this heretofore unrecoverable remaining oil amounts to about 300 billion barrels in the U.S. alone (see Figure 1), one can see why enhanced recovery is getting so much attention.

Among the broad class of enhanced oil recovery processes are processes called chemical flooding. These chemical flooding processes are extremely complex and really not completely understood. For example, a chemist in the laboratory can design a process which will recover 80 to 100 percent of the oil-in-place from a representative piece of reservoir rock; however, when these same systems are applied in the field, recoveries have ranged from 15 to 65 percent. This implies that there are interactions which sometimes occur in the reservoir that are sufficiently off design relative to laboratory work, to result in near or total process failure. To this technical uncertainty, add the problems of selecting a specific process for a given reservoir and the tremendous economic uncertainties that still exist; we see why chemical flooding is not projected to recover substantial amounts of oil for this country by the year 1985.

One way to accelerate the amount of oil recovery from chemical flooding is to obtain a better understanding of what takes place in a chemical flooding project. This understanding can be achieved through the interaction between mathematical models, laboratory results and actual field experiments. Much work is going on in the laboratory, and much work is going on in the field. However, until very recently there did not exist a mathematical model which could be used to better evaluate and understand the complex physics involved in all these processes and be used to scale the laboratory results to the field. INTERCOMP has developed such a chemical flooding model, and the development and application of this model is the subject of this paper.

The term "chemical flooding" used here is only one of many names used by the oil industry to describe similar oil recovery processes. Other names include micellar-polymer flooding, low tension waterflooding and soluble oil flooding, as well as several trademarked names for processes developed by specific companies. These have in common the injection of a chemical, normally a surfactant (or soap) to reduce the forces that trap oil in the rock in the presence of water. With these forces reduced, the oil can be mobilized and driven to a production well. Although the process is simple in concept, the specific interactions of the fluids involved are quite complex.

In order to model the behavior of chemical flooding, a number of primitive unknowns need to be identified and the complex partial differential equations which describe mass balances on these unknowns need to be solved in one, two and three dimensions. This, however, is not all of the problem. Because of the important physical processes which are taking place, the mathematical solution of these equations needs to be far more accurate in some instances than in any other petroleum recovery technique which is currently being routinely modeled. This paper presents results of some approximations which have been successfully used by INTERCOMP to generate solutions with acceptable accuracy.

This model is currently being used in an application to help engineer a micellar-polymer flood in the Bell Creek field for Gary Operating Company. The model has been effectively used to characterize two quite different chemical flooding processes and has aided Gary in selecting the process which will be used in the field. A discussion of the selection process and some preliminary engineering results which should enhance the performance of Gary's field pilot are also presented. While the discussion of the Bell Creek Project is very preliminary, we believe that this is the beginning of an understanding which will ultimately lead to much better agreement between project design and field performance. If this goal is achieved, we will be well on our way to knowing how many of the 300 billion barrels of now unrecoverable oil can be recovered economically.

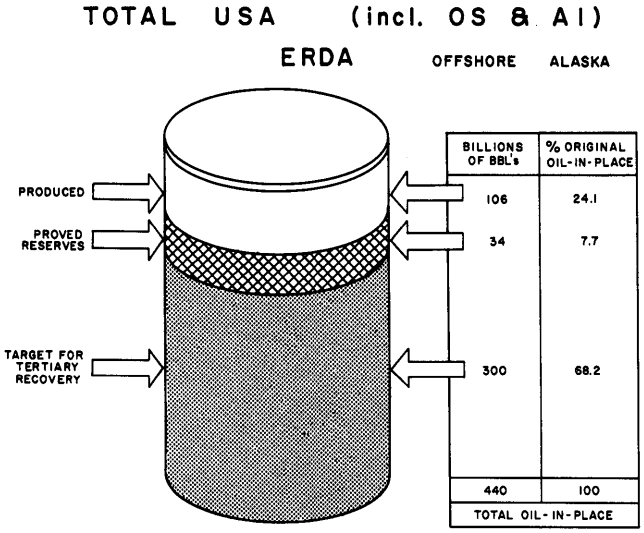


Figure 1

SPECIFIC TREATMENT

Chemical flooding is the name we shall use to denote these reservoir processes characterized by the injection of an agent to reduce the surface tension between oil and water and, hence, allow the displacement and recovery of oil that is normally trapped by capillary forces as a waterflood residual. The process is known alternatively throughout the industry as miscible-type waterflooding,[1] micellar flooding,[2,3] low tension waterflooding,[4] surfactant flooding,[5–7] and soluble oil flooding.[8]

Although chemical flooding could be applied for secondary as well as a tertiary oil recovery, the typical chemical flooding candidate is currently a watered-out oil zone; the in situ water being a normal oil field brine, high in total dissolved solids and divalent cations. The chemical flooding process thus consists of (1) a brine preflush to condition the formation and provide a controlled fluid environment that will allow optimum activity of the following surfactant system; (2) a small slug of a surfactant fluid, generally consisting of a dilute concentration of petroleum sulfonate in brine, with the possible additions of cosurfactants, polymers and other chemicals to stabilize the system, enhance the surfactant activity, reduce adsorption losses and control slug mobility; (3) a mobility control slug consisting of a dilute solution of polymer in brine, used to protect against backside dilution or overrunning of the surfactant slug by drivewater and to enhance areal and vertical sweep efficiency; and (4) a waterdrive to sweep the displaced oil, water and injected fluids to the producers.

As pointed out by Knight et al.,[15] use of a brine preflush is not universal and certainly there are nuances that are unique to each company's process. However, the above description appears to be the generally accepted characterization of the chemical flooding process.

The fluid characteristics and transport mechanisms important to the chemical flooding process have been studied extensively throughout the industry. A few of these studies are described in References 1-14 and we have included in our model the mechanisms and fluid properties indicated to be significant.

The chemical flooding system can be represented by six species or components flowing in a maximum of two phases. We may think of species one through six as water, oil, polymer, surfactant, salt and divalent ions, and denote the phases as aqueous and hydrocarbon. However, if we write the conservation equations for species three through six as to allow for (a) partitioning between aqueous and hydrocarbon phases, (b) adsorption on the reservoir rock, and (c) bulk transport and dispersion in both phases, other systems may be examined as well. In fact, the formulation used should allow the study of all of the mechanisms which have been indicated by the literature to be important.

Model formulation

The chemical flooding model solves for six components (or species) in two fluid phases. All six components may partition amongst the aqueous and hydrocarbon phases. Four components may disperse by both molecular diffusion and mechanical dispersion in both the longitudinal and transverse direction and in both phases. The same four components may adsorb on the reservoir rock, satisfying equilibrium sorption isotherms.

The primitive unknowns to be solved for by the simulator are:

- WA1 Mass fraction of water in the aqueous phase
- WH2 Mass fraction of oil in the hydrocarbon phase
- WA3 Mass fraction of component three (normally polymer) in the aqueous phase
- WA4 Mass fraction of component four (normally surfactant) in the aqueous phase
- WA5 Mass fraction of component five (normally monovalent salt) in the aqueous phase
- WA6 Mass fraction of component six (normally divalent salt) in the phase saturation
- SA Aqueous phase saturation
- p Aqueous phase pressure

The equations to be solved by the simulator are:

Component 1 (Water)

$$\frac{\partial}{\partial t}(\phi S_a \rho_a WA1 + \phi S_h \rho_h SH1) + \nabla\cdot(\rho_a WA1 \tilde{U}_a + \rho_h WH1 \tilde{U}_h) - \tilde{q}_1 = 0 \quad (1)$$

Component 2 (Oil)

$$\frac{\partial}{\partial t}(\phi S_a \rho_a WA2 + \phi S_h \rho_h WH2) + \nabla\cdot(\rho_a WA2 \tilde{U}_a + \rho_h WH2 \tilde{U}_h) - \tilde{q}_2 = 0 \quad (2)$$

Component 3 (Polymer)

$$\frac{\partial}{\partial t}(\phi_p S_a \rho_a WA3 + \phi_p S_h \rho_h WH3 + \phi \rho_{r3}) + \nabla\cdot(\rho_a WA3 \tilde{U}_a + \rho_h WH3 \tilde{U}_h) - \nabla\cdot(\phi_p S_a \rho_a \underline{\underline{K}}_{a3} \nabla WA3 + \phi_p S_h \rho_h \underline{\underline{K}}_{h3} \nabla WH3) - \tilde{q}_3 = 0 \quad (3)$$

Components 4 (surfactant), 5 (monovalent salt) and 6 (divalent salt) equations are the same as that for Component 3, except that ϕ_p is replaced by ϕ.

There are three constraint equations for the system. The first requires the sum of saturations be equal to one. This equation is satisfied by setting $S_h=1-S_a$ in a six component balance equations. The other two constraint equations require that the sum of the mass fractions in each phase equal one. These equations are left explicitly in the formation as Equations (7) and (8). By ordering the unknowns as shown on the list of primitive unknowns, Equations (7) and (8) eventually become the "saturation" and "pressure" equations.

Assuming an isothermal system with compositions independent of pressure, we have the following equilibrium relationships for partitioning of species one through six between the aqueous and hydrocarbon phases and for adsorption of species three through six on the reservoir rock:

$$WHi=f_i(WAj)_{j=1-6} \quad \text{for i=1 through 6} \tag{9}$$

$$\rho_{Ri}=f_{Ri}(WAj)_{j=1-6} \quad \text{for i=3 through 6} \tag{10}$$

For equations of state, we have

$$\rho_a=\rho_a(WAi,p_a)_{i=1-6} \tag{11}$$

$$\rho_h=\rho_h(WHi,p_h)_{i=1-6} \tag{12}$$

$$\tilde{U}_a=-\frac{k_a}{\mu_a}(\nabla p_a-\rho_{ae}g\nabla D) \tag{13}$$

$$\tilde{U}_h=-\frac{k_h}{\mu_h}(\nabla p_h-\rho_{he}g\nabla D) \tag{14}$$

where

$$k_a=k_a(S_a,WAi)_{i=1-6} \tag{15}$$

$$k_h=k_h(S_a,WAi,\tilde{U}_a)_{i=1-6} \tag{16}$$

$$\mu_a=\mu_a(WAi,S_a,\tilde{U}_a)_{i=1-6} \tag{17}$$

$$\mu_h=\mu_h(WAi,S_a)_{i=1-6} \tag{18}$$

Assuming that the capillary pressure is a function of both saturation and composition, we have finally

$$p_h=p_a+P_c(S_a,WAi)_{i=1-6} \tag{19}$$

MODEL SOLUTION

As if developing numerical solutions to Equations (1) through (19) is not complex enough, the numerical solution needs to be extremely accurate in order to be able to model the phenomena of dispersion and fingering. If it weren't for these special problems, the numerical solution to Equations (1) through (19) could follow any normal finite difference or finite element type approximation, and for this reason will not be addressed in this paper. We will, however, address our attention at the specific problems of dispersion and fingering since these are truly important physical phenomena to be modeled and help to make some of the interesting points of this paper.

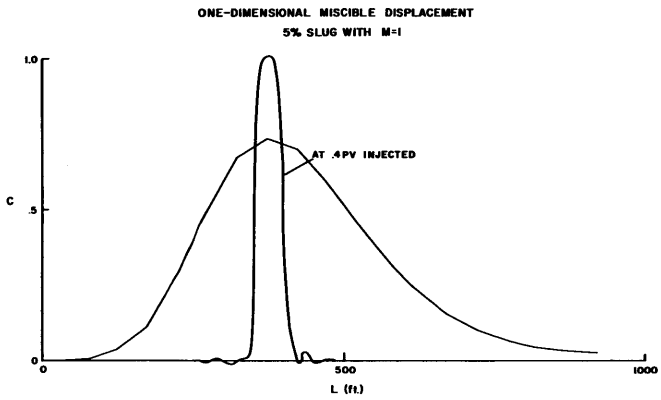


Figure 2

Figure 2 is included here to illustrate the effects of numerical dispersion. This figure displays the results of a simple, one-dimensional calculation. If the solution were infinitely accurate, the high spike shown on this figure would result from the effects of actual physical dispersion on the solution. As you can also see on Figure 2, the very smeared curve displays the effects of large numerical truncation error which has the exact same effect as a large physical dispersion. Figure 3 is an illustration depicting the phenomenon of viscous fingering. This occurs whenever a more mobile fluid is injected into a less mobile fluid; the more mobile fluid wants to finger through the less mobile fluid to the producing well providing a thief zone for the injected fluid. This phenomenon results in cycling of the injected fluid without really displacing the fluid in place. In chemical flooding, because the costs of the chemicals injected are so extremely high, all of the processes are designed around putting in as little chemical in the ground as is absolutely necessary.

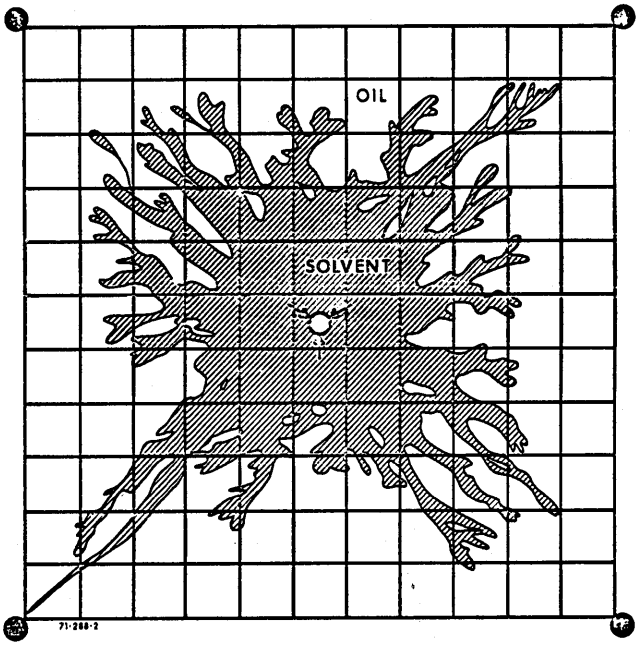
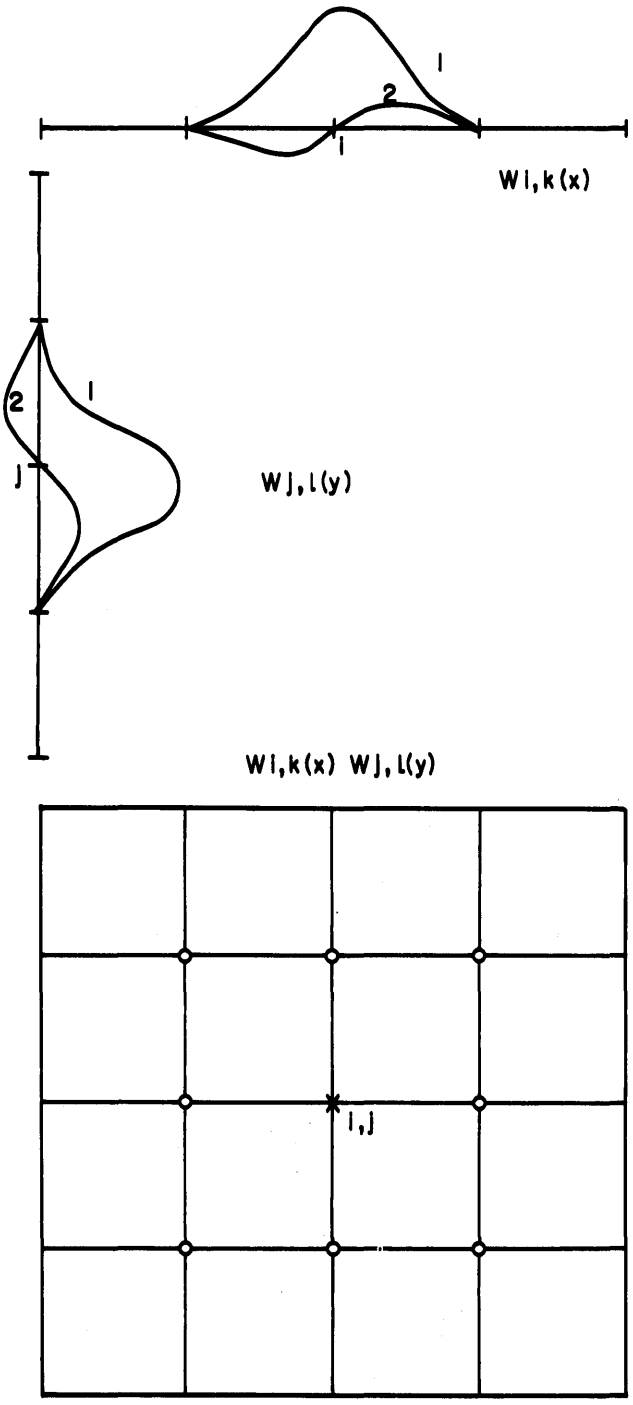


Figure 3



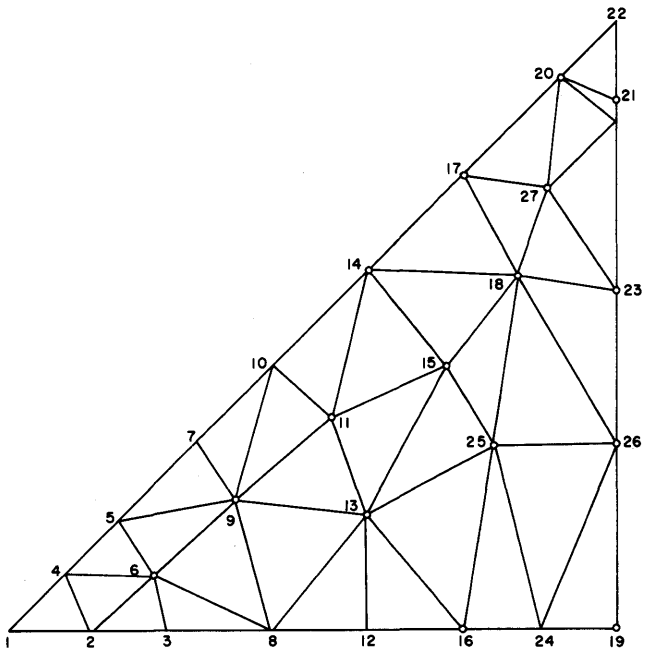
HERMITE PIECEWISE CUBIC ELEMENTS

Figure 4

Because of this, the effects of both physical dispersion and viscous fingering are extremely important in the field and need to be modeled accurately if realistic results are to be obtained.

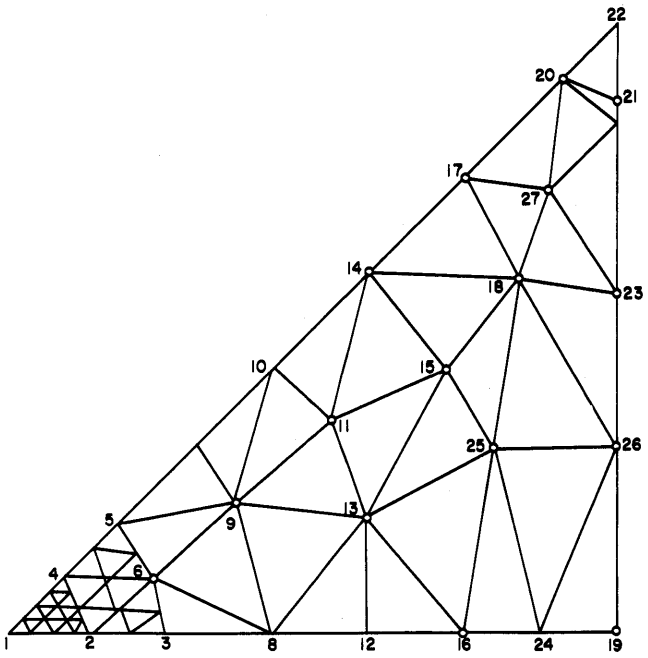
There are two approaches to solving partial differential equations describing the chemical flooding process and still be able to achieve the kind of accuracies required in order to model dispersion and fingering. The first approach is to use high order finite element approximations and the second approach is to use some local grid modification such that the accuracy can be concentrated in the areas where it is required and coarse grids can be used elsewhere. The high order finite element approximations are depicted by Figure 4 where I have illustrated here third order, cubic Hermite polynomials which can very effectively be used to generate fourth order accurate approximations. Figures 5, 6 and 7 illustrate a specific type of local grid refinement. Here we have an original triangle grid in Figure 5 and two levels of refinement shown right at the injection well which is in the lower left-hand corner of Figure 6. Figure 7 illustrates a later point in time when this refinement has moved in somewhat away from the injection well in order to accurately describe a front which is moving away from the injection well. This is a local refinement that needs to be time varying and move along with the front.

To illustrate the importance of these phenomena, I've shown on Figure 8 the effects of dispersion on recovery. You will note that as the physical dispersion is reduced, the peak concentration of the injected slug is increased and, therefore, the amount of oil recovered by continuous displacement of the injected fluid is significantly increased. Figure 9 is an illustration of a computer plot which shows a first order approximation to the injection of a slug of displacing fluid at a one-to-one mobility ratio. You will note how dispersed this particular slug is relative to Figure 10, which represents a fourth order correct solution to this problem. You will note also Figure 11 where the use of local grid refinement has been used how much sharper and higher the



MACRO TRIANGULATION

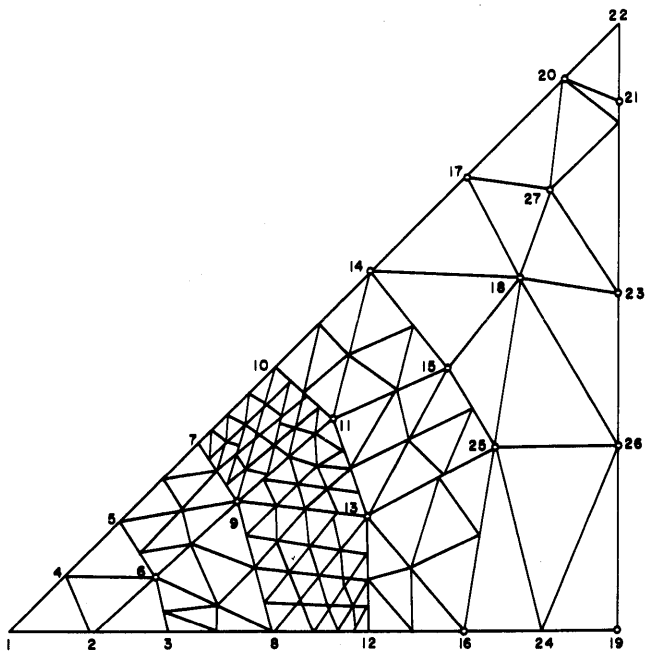
Figure 5



MACRO TRIANGULATION

LOCAL REFINEMENT

Figure 6



MACRO TRIANGULATION

LOCAL REFINEMENT

Figure 7

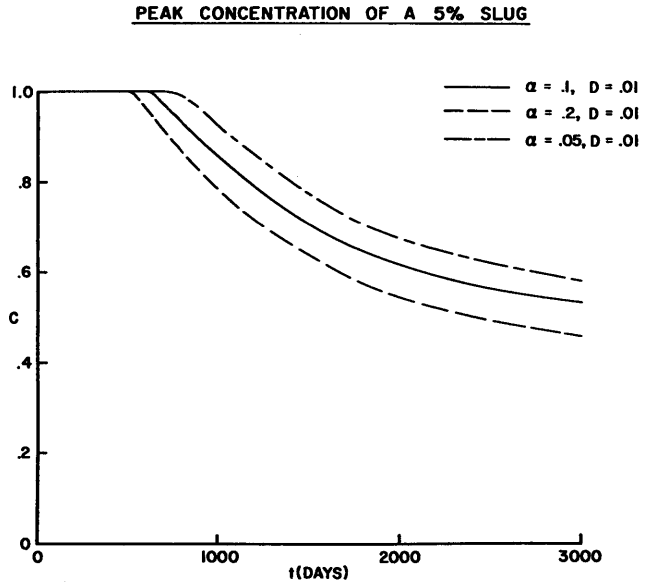


Figure 8

peak slug concentrations are than in Figure 9. Figure 12 is an illustration of a fourth order finite element approximation's ability to actually model viscous fingering in an unfavorable mobility ratio displacement and Figure 13 is an illustration of how the model is able to accurately predict recoveries at low dispersion levels for unfavorable mobility ratio displacements by introducing nominal heterogeneities into the system to propagate viscous fingers.

One further phenomenon which results when finite difference approximations are used to model unfavorable mobility ratio displacements is that of grid orientation. Figure 14

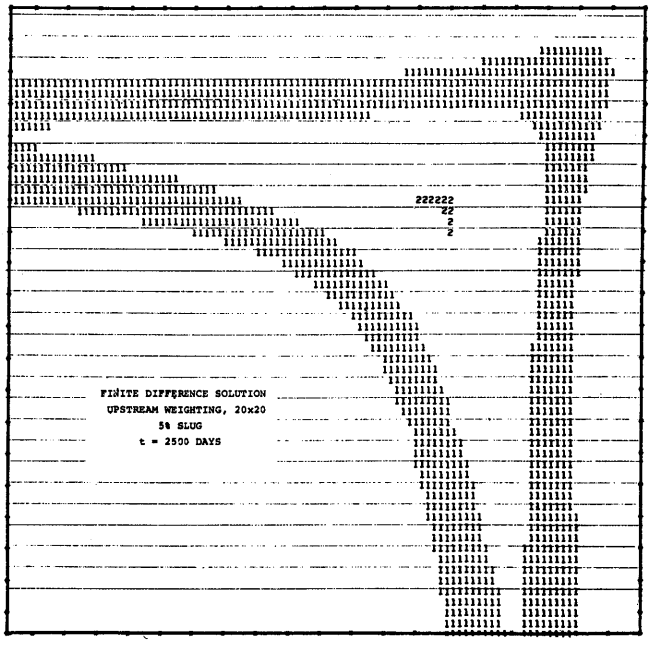


Figure 9

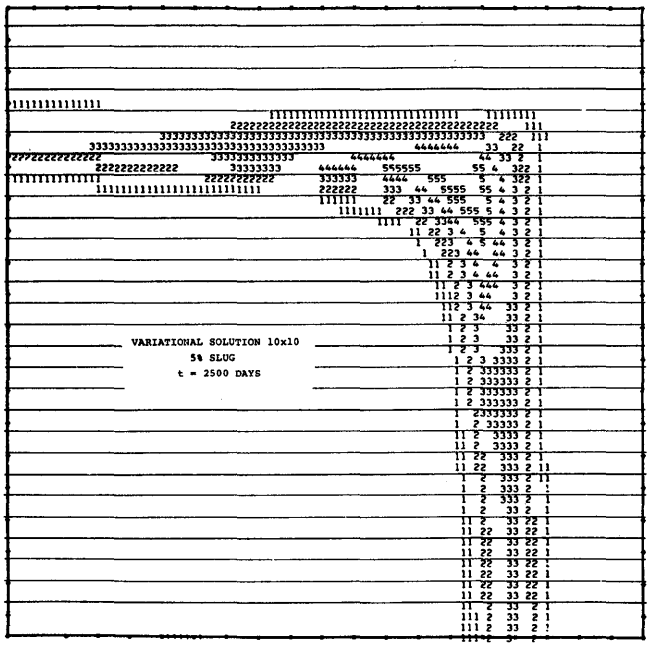


Figure 10

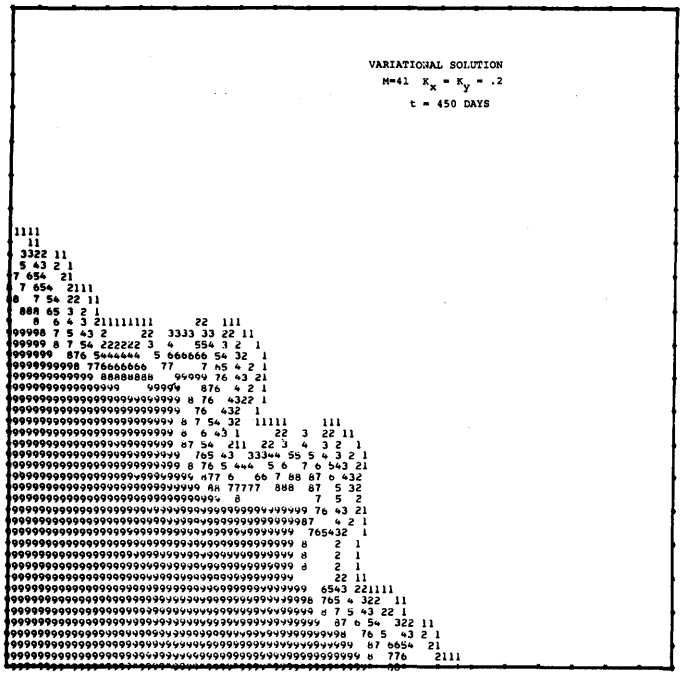


Figure 12

shows a repeated five-spot pattern and illustrates the two different grids that could be used to represent the flow of fluids in a five-spot. You will note that if the regions are totally homogeneous and if all the injection wells have the same rates and all the producing wells have the same rates, each of these repeated patterns should perform identical to

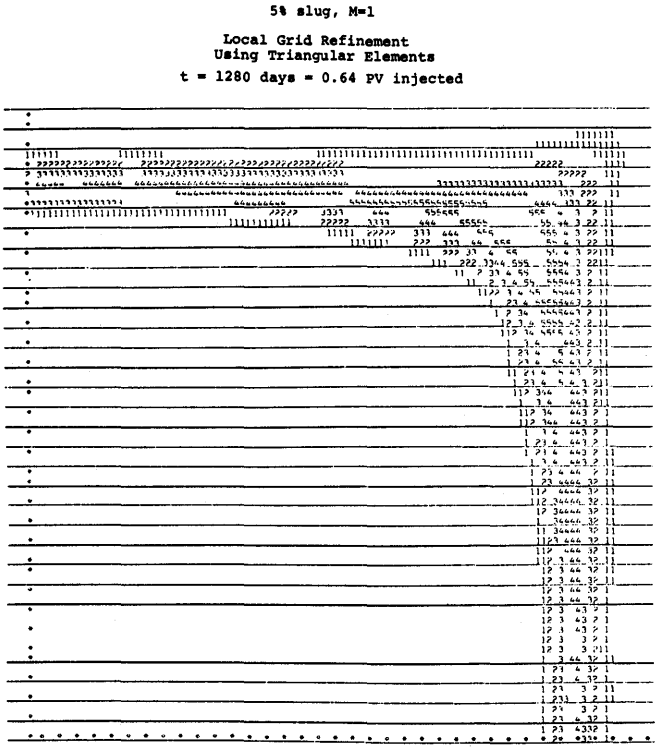


Figure 11

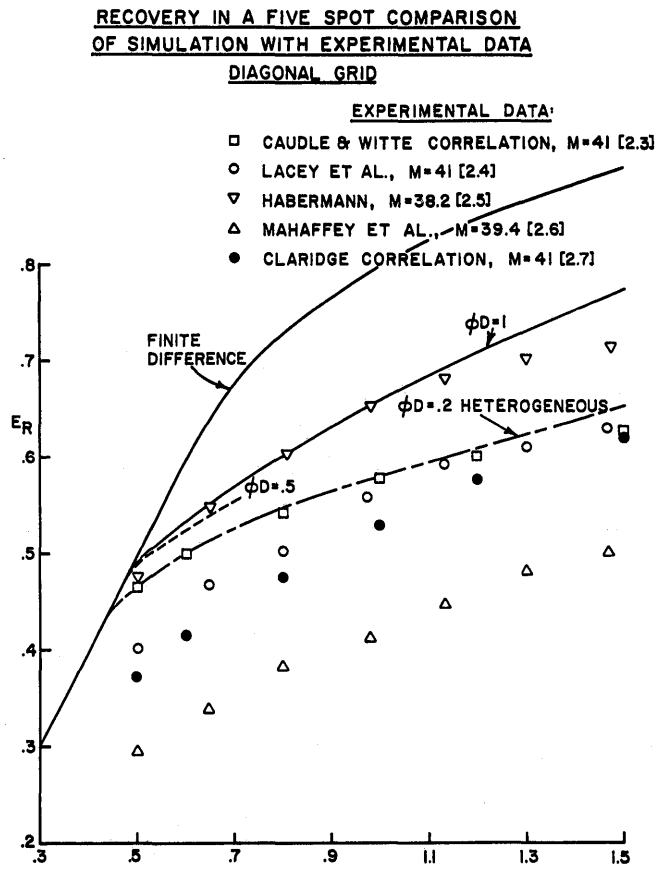
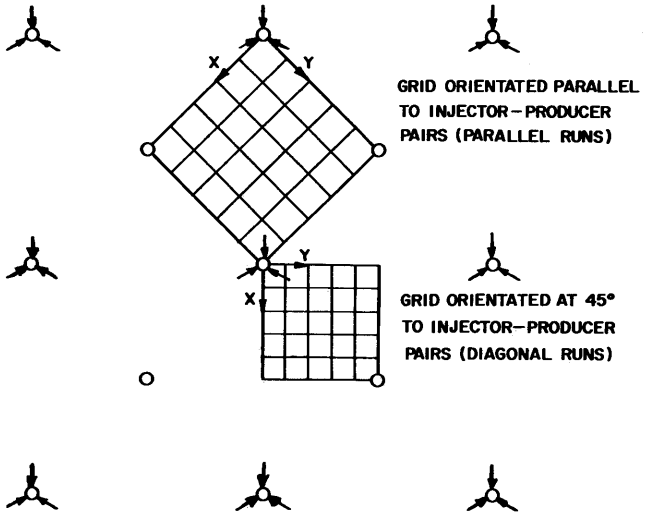


Figure 13



GRID ORIENTATION FOR SIMULATION OF DISPLACEMENTS IN FIVE-SPOT SYMMETRY ELEMENTS

Figure 14

every other repeated pattern. You will note from this figure that one of the grids is parallel to the coordinate axes while the other grid is running at an angle of 45° to the coordinate axes. These two grids are called parallel and diagonal for the sake of discussion. You will note in Figure 15 how the recovery performance for a very simple displacement process can vary depending on whether one uses the parallel or the diagonal grid. As mentioned before, these results should be identical and yet there is as much as a 20 percent difference in the recovery of oil after one pore volume of fluid injected. This is clearly an intolerable error and needs to be eliminated. It can also be seen on this same curve that high order finite element approximations do not exhibit this grid orientation phenomenon. In fact, none of the finite element approximations exhibit grid orientation effects which plague finite difference schemes.

We have illustrated in this section how the problem of describing a physical process such as chemical flooding requires numerical approximations which not only solve complex partial differential equation with complex nonlinearities, but which also must describe the important physics of the problem. Finite element approximations are definitely recommended when high order accuracy to model effects such as physical dispersion and/or viscous fingering are required.

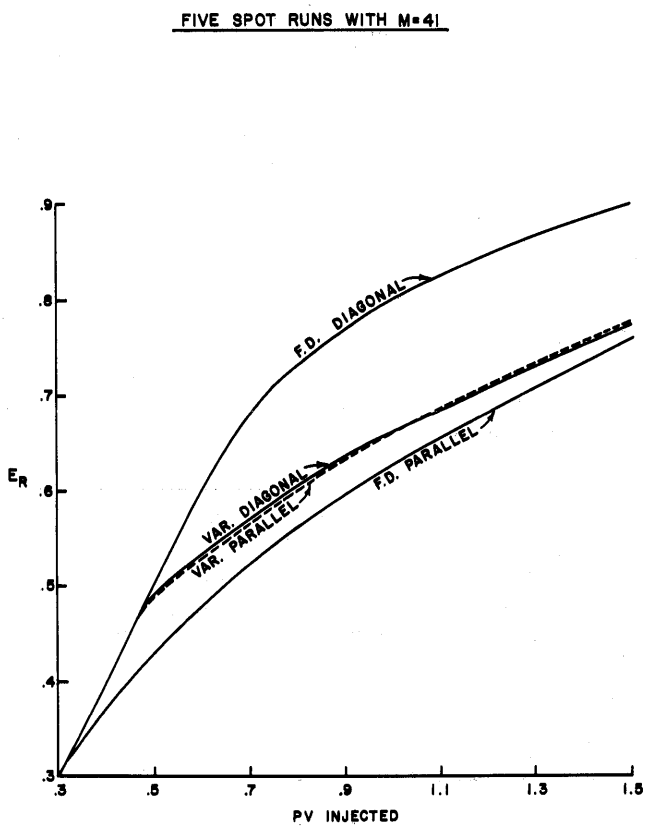


Figure 15

MODEL APPLICATION

Once a complex physical model, such as the one described by Equations (1) through (19), has been coded and debugged and running smoothly on a large-scale digital computer, one of the first things that needs to be done is to validate the model against actual physical processes in order to determine that the model does in fact describe the complex physics that it was intended to describe. This is really the stage of model development in which INTERCOMP is currently engaged; however, we have modeled two complex chemical flooding schemes and will illustrate here how the model was utilized in these particular instances.

The actual study that was performed was to model two competing design philosophies and to utilize the model to determine which of these should be implemented in Gary Operating Company's Bell Creek micellar-polymer pilot. The two chemical design philosophies which were modeled were (1) a small slug high concentration "soluble oil" system, and (2) a large slug dilute concentration, "optimal salinity" design.

A number of laboratory experiments were run in order to obtain basic data for the model as well as to test the design philosophies in displacing oil from the Brea core. While a large number of experiments were actually modeled, I have chosen just two here to illustrate the actual capabilities of the chemical flood program. You will note from Figure 16 how good an actual match was obtained for a .09 pore volume slug using the high water content "HWC" process and from Figure 17 we have illustrated the match for the soluble oil process using a .03 pore volume slug. The interesting features about these history matches were that both

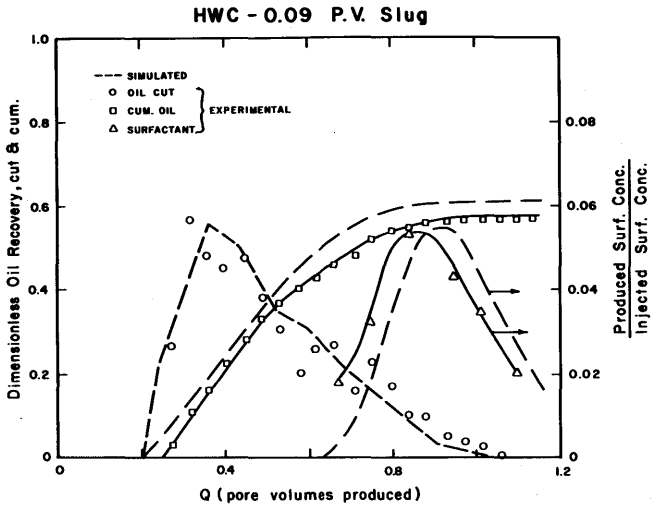


Figure 16

model runs utilized basically the same data and that only the different description of the physics resulted in the different model predictions. While these history matches are not perfect, they could easily have been further adjusted and fine tuned. However, it was not felt that this was necessary since the purpose here was really to demonstrate the model's capability to describe the physics of the two completely different systems. Once the history match was obtained then the model was used to predict the ultimate recovery from the cores using various weight percent surfactant concentrations. You see from Figure 18 that here again the model did an excellent job of reproducing laboratory experiments and that clearly the soluble oil system, at least in the core, was significantly better than the high water content process.

Once this history match was obtained the model was then utilized to make both areal and vertical cross section runs to see how these processes might deteriorate in the field. It

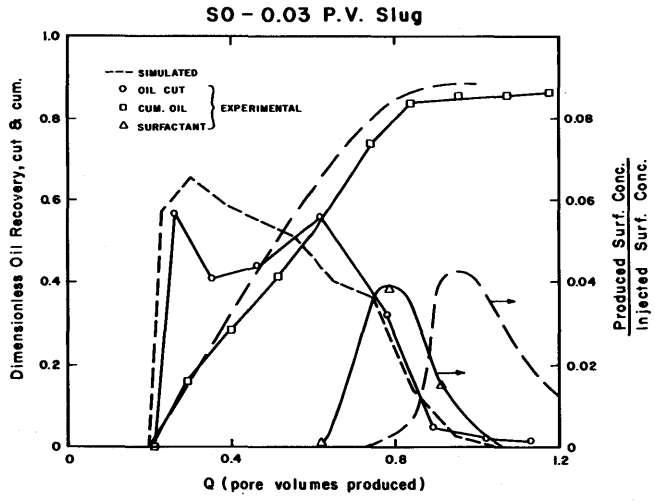


Figure 17

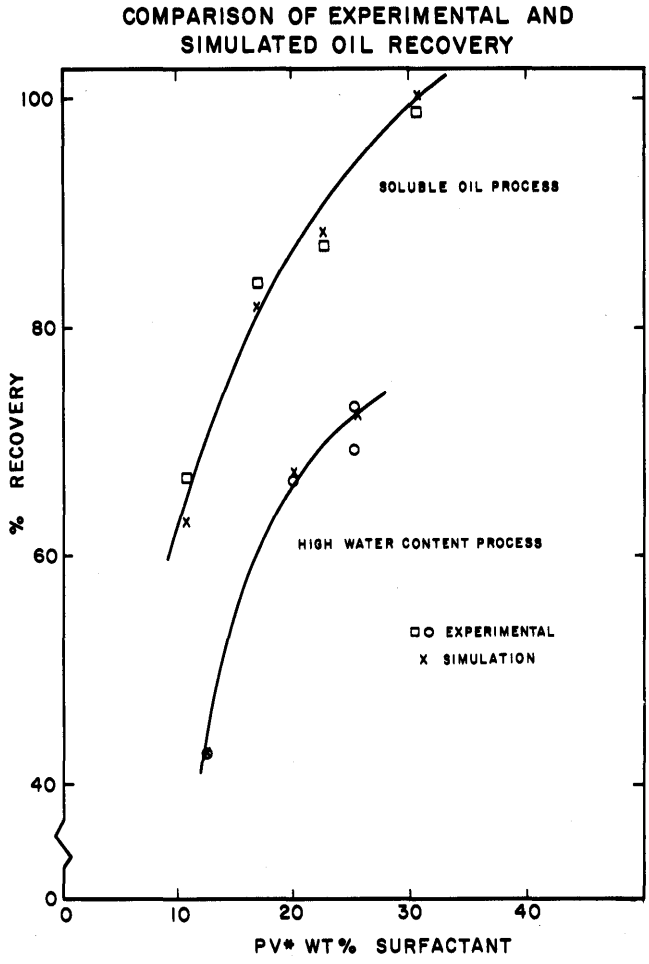


Figure 18

is interesting to note that once we made these runs and combined the areal and the vertical runs to estimate what would be recovered from the pilot that the model predicted a 55 percent recovery with a soluble oil process and only a 35 percent recovery with a high water content process. These results are still preliminary since we did not at this point look at any degradation due to layering for these systems. However, it is very interesting to note that a process which recovers almost 90 percent of the oil in a core would be predicted to recover only 55 percent of the oil in an idealized pattern. This is far more like what is actually being observed in the field and is extremely encouraging that we have now reached the point where mathematical models can be used to predict oil recovery by chemical flooding.

Again while this work is still very preliminary, I would like to point out one other very interesting thing that has been learned to date from this particular study. The field pilot has been designed as a five-spot with this expanded to the full field originally if the pilot is successful. However one of the interesting things that was noted in this work is that because of the diverging nature of the flow in a five-spot as the slug moves away from the injection well very

little oil is displaced around the edges of the pattern. In fact, almost no oil is being mobilized around the edges and most of the oil being recovered from the pilot is coming right out of the stream tubes that run between the injector and the producer. This gives the strong indication that line drives or patterns other than five-spots might be considerably better for recovering oil via chemical flooding. This work will be expanded upon and may in fact even result in some design modification for this very exciting and interesting pilot project.

CONCLUSIONS

In this paper we have presented some of the interesting aspects in the development and application of a chemical flood model. This clearly illustrates how computers and mathematical models can play an extremely important role in aiding the oil industry in recovering additional oil. The complex model which was developed is now being utilized to better understand laboratory work which will in turn enable us to scale the very important laboratory experiments to actual field scale projects and hopefully enable us to obtain a better understanding why some processes seem to work and other processes seem to fail. Once a process is characterized and the reservoir is characterized we feel that a mathematical model of this sort is an excellent predictive tool and should go a long way toward enabling the oil industry to recover a substantial fraction of this 300 billion barrels of oil which is left in the ground and currently called unrecoverable.

REFERENCES

1. Gogarty, W. B. and W. C. Tasch, "Miscible-Type Waterflooding: Oil Recovery with Micellar Solutions," *J. Pet. Tech.*, December 1968, pp. 1407-1414.
2. Trushenski, S. P., D. L. Dauben and D. R. Parrish, "Micellar Flooding-Fluid Propagation, Interaction and Mobility," *SPEJ*, December 1974, pp. 633-642.
3. Gogarty, W. B. and H. Surkalo, "A Field Test of Micellar Solution Flooding," *J. Pet. Tech.*, September 1972, pp. 1161-1169.
4. Foster, W. R., "A Low-Tension Waterflooding Process," *J. Pet. Tech.*, February 1973, pp. 205-210.
5. Hill, H. J., J. Reisberg and G. L. Stegemeier, "Aqueous Surfactant Systems for Oil Recovery," *J. Pet. Tech.*, February 1973, pp. 186-194.
6. Gale, W. W. and E. I. Sandvik, "Tertiary Surfactant Flooding: Petroleum Sulfonate Composition-Efficacy Studies," *SPEJ*, August 1973, pp. 191-199.
7. Gilliland, H. E. and F. R. Conley, "Surfactant Waterflooding," Panel Discussion 14(2), Preprint of the Proceedings of the 9th World Petroleum Congress, 1975.
8. Jones, S. C., "Comments on Finding the Most Profitable Slug Size," *J. Pet. Tech.*, January 1973, pp. 77-79.
9. Paul, G. W. and H. R. Froning, "Salinity Effects of Micellar Flooding," *J. Pet. Tech.*, August 1973, pp. 957-958.
10. Mungan, N., "Shear Viscosities of Sonic Polyacrylamide Solutions," *SPEJ*, December 1972, pp. 469-473.
11. Hirasaki, G. J. and G. A. Pipe, "Analysis of Factors Influencing Mobility and Adsorption in the Flow of Polymer Solution through Porous Media," *SPEJ*, August 1974, pp. 337-346.
12. Slobod, R. L. and S. J. Lesty, "Use of Graded Viscosity Zone to Reduce Fingering in Miscible Phase Displacements," *Producers Monthly*, August 1960, pp. 12-16.
13. Bondor, P. L., G. J. Hirasaki and M. J. Tham, "Mathematical Simulation of Polymer Flooding in Complex Reservoirs," *SPEJ*, October 1972, pp. 369-382.
14. Dawson, R. and R. B. Lantz, "Inaccessible Pore Volume in Polymer Flooding," *SPEJ*, October 1972, pp. 448-452.
15. Knight, B. L., S. C. Jones and R. W. Parsons, "Discussion (Reference 2 above)," *SPEJ*, December 1974, pp. 643-644.

Geophysical DP requirements could exceed the world's GP capacity by 1985

by CARL H. SAVIT

Western Geophysical Company of America
Houston, Texas

INTRODUCTION

The reflection seismic method of underground mapping is today the only practicable means of selecting a place to drill a new oil or gas well. As a consequence, the free world spends about two billion dollars a year for reflection seismic exploration.

All of this activity consists of data gathering, data reduction, and data analysis. In 1977 the industry recorded between 10^{14} and 10^{15} bits of primary data in digital form. All of these data was processed at least once. Indeed, it is probable that the geophysics industry processes more primary digital data than does any other activity. Worldwide, the industry dedicates computer power equivalent to a few dozen IBM 370/168's to processing seismic reflection data.

During 1977 new field techniques were introduced to improve the resolution of the entire process. These new techniques begin by recording about 20 times as much data per unit of survey as are being recorded in conventional, present-day practice. A further increase to about 50 times the present data rates will be needed to make the horizontal resolution of the system reasonably consistent with the vertical. Ultimately, algorithms that take full advantage of the increased recording density will require 1000 or more additional computation steps per input point. Consequently, we perceive a need for a 50-fold increase in internal memory and a 50,000-fold increase in arithmetic speed. In terms of "power" (where power is defined as the product of internal memory size and the number of operations per unit time), these requirements translate to a 3×10^6-fold increase.

It has been the experience of the past three decades that available computer power (as we have defined it) increases about one order of magnitude every 2.7 years. If, as seems reasonable today, this trend continues until 1985, our computers will then be about 10^3 times as powerful as they were in 1977. Also, as seems reasonable today, reflection seismologists will by 1985 be ready for the 3×10^6-fold increase we have projected. To accomplish the same number of units of survey in 1985 as were done in 1977, we could be facing a requirement for 3000 times as many 1985 model computers as the number of 1977 model machines we used in 1977. It would, in addition, be reasonable to expect the market for surveys to have doubled in the next seven years so that we may need to increase our demand by an additional factor of two. (Figure 1)

DATA AND ALGORITHMS

To understand the nature of the geophysical computation requirement, an overview of the basic physics of reflection seismology will be of some help.

In reflection seismology a mechanical excitation is applied at or near the surface of the earth. Pick-ups also on or near the surface and relatively near the point of excitation detect the response of the earth to the excitation. From the complex pattern of energy returned to the surface, the exploration geophysicist attempts to select the longitudinal elastic waves (i.e., the sound waves) that have returned to the surface by reflection from underground interfaces between different rock layers. In the current state of the art, energy of all other kinds is rejected or suppressed as unwanted "noise."

Nearly all processing methods in general use into the first few years of this decade were based on the concepts of geometrical optics. In other words, sound waves were assumed to travel along rays and to be reflected or refracted at the point of intersection of each ray with an interface or discontinuity. (Figure 2)

As a consequence of these assumptions, deriving a representation of the configuration of underground layering from the received acoustic signals involves the use of little more than the algebraic and trigonometric equations of analytic geometry. One major intermediate problem, that of deriving the velocity of sound as a function of depth from the surface, appears to require the solution of an integral equation. Instead, however, of attempting an analytic solution in the face of noisy and incomplete data, the exploration seismologist has opted for the method of exhaustive search. He solves the forward problem of determining a suite of likely reflection times for selected parts of the data over an anticipated range of assumed velocities. The algebra required is primarily the Pythogorean equation followed by a vast number of correlation calculations to determine the degree to which reflection times derived from the searched velocities fit the observed signals.

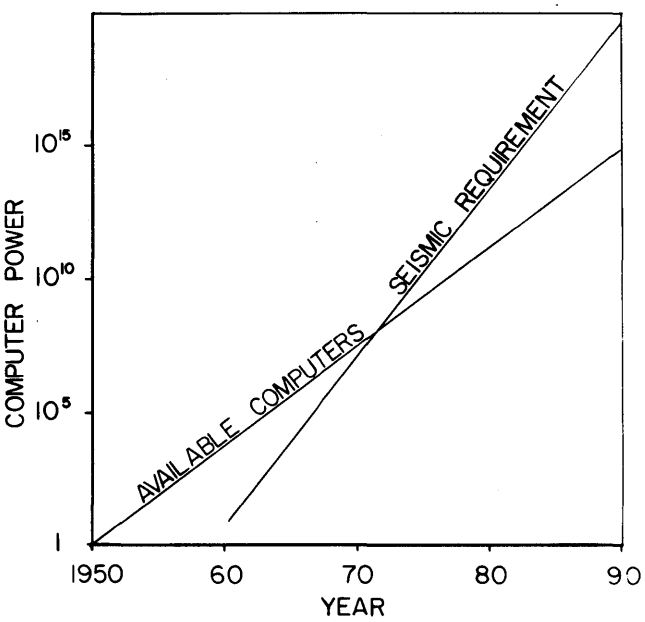


Figure 1—The power of typical commercially available computers has been growing about one order of magnitude every 2.7 years and should continue to do so through the 1980's. Requirements of the seismic exploration industry have been growing and should continue to grow at a substantially faster rate.

The necessary ray-path calculations are quite simple and so produce a relatively small computational load. Most of the computation time is devoted to the extraction of good acoustic "signals" from noisy data. The two approaches used to improve signal-to-noise ratios are the averaging of redundant data and filtering by means of a battery of sophisticated techniques.

By far the largest computing load stems directly from the high redundancy of data being taken. A typical field record of the response of the earth to one mechanical input consists of 3000 data values taken simultaneously at each of 48 equally spaced surface locations. Each input thus produces about 150,000 floating-point numbers. Floating point is required because the signal immediately after the instant of input can be four orders of magnitude larger than the signal six seconds later when the recording is finished.

Typically one signal input is applied every 30 to 60 meters along a line of survey so that a seismic survey produces

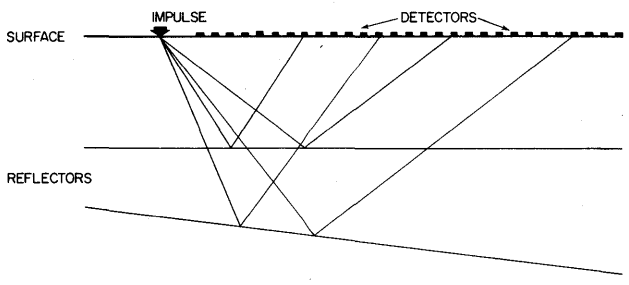


Figure 2—The geometric-optic model of reflection seismology postulates a unique raypath connecting an impulse, a given reflection, and an individual detector; and the angle of incidence is equal to the angle of reflection.

some 2.5×10^6 to 5.0×10^6 floating-point data values per kilometer.

Such surveys produce data redundancies ranging from 12 to 24 or 48. When more redundancy is required to achieve useful anti-noise discrimination, more data are taken per kilometer of survey. Redundancies of 96 are not uncommon in special situations.

In nearly half of all surveys on land, the input consists of a swept-frequency (chirp) signal of substantial duration. Systems using this form of input require the recording of thirty times more data values per kilometer. Simple computation in the field compresses these data about 16-fold, but the result of this compression must be further processed before it reaches the computational level of unprocessed impulse data.

Before redundant data can be combined or averaged, several distinct data processing steps must be applied. Those redundant data that apply to, or represent, the same subsurface point are gathered from diverse input and pick-up points. The different sound waves reflected from the same subsurface point have travelled paths of different length and have encountered different inhomogeneities in their paths. Field recordings are, of necessity, made in real time so that data from different pick-ups must be recorded in time-multiplexed order.

Normally, the first computational step is to demultiplex the data into time-sequential value sets for each pick-up. Next, data are normalized both as to amplitude and time scale to approximate values that would have been produced by horizontal plane waves. The necessary transformations are non-linear but have simple geometric bases.

In most cases the first of several filtering steps is applied to the data after the amplitude normalization and before application of the time-scale normalization. Almost always the filter is in the nature of a deconvolution; that is, an attempt is made to undo the various spread functions that characterize the effective shape of the input signal. That the input function has substantial duration is the result both of the imperfect nature of the mechanical impulse itself and of the reverberatory character of the earth materials near the input source.

Designing the filter operators to be applied to the data to convert the input signal to a band-limited, simple impulse function represents a substantial computation load. While it is a relatively simple matter to use the Wiener-Levinson algorithm to invert a known (i.e., recorded-in-the-field) mechanical impulse shape to a desired simple impulse, it is somewhat more burdensome to derive the analogous operator to compensate for an unknown reverberation function. The usual method is to derive the amplitude spectrum of the reverberation function from autocorrelations of the unfiltered data and to assume (on physical grounds) that the function is minimum-phase. Other more complex schemes are in use to various extents.

All of the steps described to this point are necessarily applied to essentially all of the field-recorded data and, hence, represent the great bulk of the computation load. In those cases in which the velocity of seismic waves varies rapidly along the line of survey, the intermediate step of

velocity determination, described earlier, can dominate the computation requirement.

In the last computer pass of these preliminary computation steps, the data are combined or averaged to produce the compacted, signal-enhanced data set for final processing. Essentially all of the redundancy has now been removed so that data volume is reduced by the original redundancy factor of 12 to 96. Furthermore, the dynamic range of the data has been sufficiently reduced that results may validly be written as 16-bit, fixed-point numbers.

Until quite recently, virtually all seismic reflection data were subjected to at most one or two additional filtering steps and were amplitude-conditioned in one of two alternative modes and formatted for presentation in graphical form as hard copy.

Such additional calculations as have been made within the geometric-optics framework have ranged from simple smoothing operations to elaborate ray-tracing routines but in the aggregate have had negligible impact on computer requirements.

ACOUSTIC WAVE EQUATION

We in the exploration-geophysics community have long been aware that the geometric-optics model of the seismic reflection process is, at best, a first approximation to reality. Even the fundamental assumption of the geometrical optics model—that the longest wavelengths involved are small compared to the dimensions of interacting elements in the medium—is at best weakly valid. Some of the wavelengths we use are indeed substantially greater than the dimensions of subsurface features of interest.

The simplest model that enables us to relax the dimensional assumption required in geometric optics is acoustical. In the acoustical model it is assumed that the propagation of seismic energy can be described by the acoustic wave equation

$$\frac{\partial^2 \varphi}{\partial t^2} = c^2 \nabla^2 \varphi.$$

That equation is the mathematical expression of a physical model in which energy is propagated in a lossless medium solely by means of variations of pressure.

It is well established that many of the defects of the ray propagation assumption of the geometric-optic model are mitigated or effectively eliminated by use of the wave-equation model. Unfortunately, these advantages are gained at the cost of orders of magnitude increases in the required computations.

This increase in computational load is inherent in the difference in the physical models. In the geometric-optic model, the reflection from a point on an interface obeys the specular rule—the angle of incidence is equal to the angle of reflection. Thus, an impulse from a particular surface point of origin can reflect at a given point on the reflector to only one receiver point on the surface. To process the information relevant to one subsurface point, it is only necessary to treat a number of data values equal to the design redundancy of the system. (Figure 2)

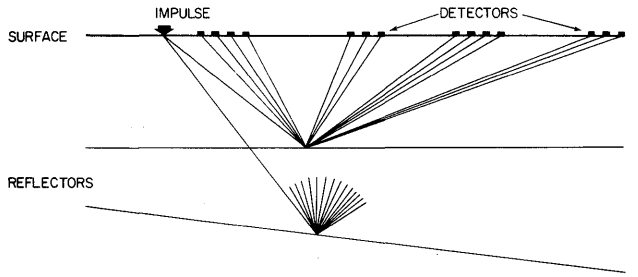


Figure 3—The acoustic model of reflection seismology postulates that energy is radiated in all directions from every point on a reflecting surface. Accordingly, *every* detector receives some energy from each reflecting point in response to a given impulse.

In the acoustic model, however, the reflection point is a new source and retransmits energy in all directions. Hence, every receiver records energy "reflected" from every point on the reflector in response to an impulse. Conceptually, we should use all data from the entire survey to determine the characteristics of each reflection point. While the details differ, operationally the process is analogous to the creation of an image corresponding to a hologram. (Figure 3)

Obviously, an immediate and complete shift from the ray-theory to the wave-equation approach would have required more computer power than is available in the entire world. In any event, the necessary analytical foundations for such a change have not yet been developed.

What has happened is that a working mix of the two models has been evolving. In its earliest manifestation the mix consisted of ray-path processing until a final compressed, enhanced data set is produced, following by a single-application of a limited-parameter version of the Kirchhoff-integral solution to the wave equation. The number crunching required for this process when applied to 48-fold-compressed data was of the same order of magnitude as that needed for all the preceding steps.

In today's world, a battery of programming techniques and analytical simplifications have made the wave-equation approach more economical and the need for better data has made funding increasingly available.

In the most sophisticated approach known to me to be in commercial use, the data are processed at least part way in the geometric-optic manner to derive model parameters which are then used for extensive acoustic-wave processing. The result of that processing is then further modified by a technique derived from geometric optics to compensate for an inadequacy in the wave-equation formulations available today. (Figure 4)

Another few years should see a fully mature acoustic-wave technique to carry the entire processing sequence, but it will almost certainly require ray-path processing as a preliminary step to determine initializing or first-approximation parameters. Much of the processing may be iterative.

For those who are inclined to worry about a possible static future, I would point out that the introduction of attenuation into the equations should suffice to keep everyone busy for some years beyond these projections. After that we could profitably spend a generation or two applying

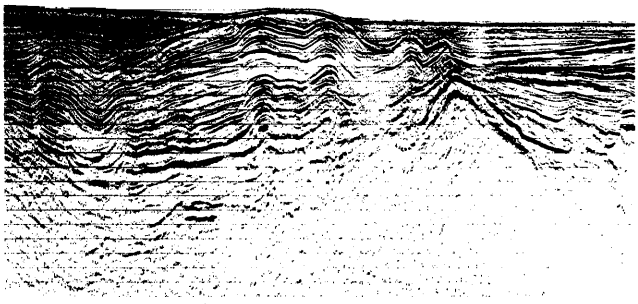


Figure 4—This cross section from offshore California represents a horizontal distance of about 26 kilometers and a total vertical dimension of 2.7 kilometers. About 3×10^9 bits of data were recorded at sea. Processing through the compaction stage was performed in accordance with the raypath model. The compacted data were treated by acoustic wave methods, and the results were then further modified by another ray-theory-derived process. Processing costs were something over $6000.

the tensor equations that describe the elastic behavior of real solids.

HARDWARE REQUIREMENTS

Since the seismic reflection industry went digital in the mid 60's, it has been pressing hard at the frontiers of computer technology. The first group to undertake regular computer processing of seismic data designed and built a special purpose system because of the limitations of the then-available GP systems. Seismic industry requirements soon stimulated the development by computer manufacturers of specialized hardware to cut computing costs for some heavily used algorithms and thus bring overall costs to an acceptable level. In the initial years, correlation and convolution operations were the pacing items making up computer costs. For that reason the first hardware specialties were a fast multiply-add instruction, followed within a matter of months by hardwired correlators. In 1966 IBM announced the 2938 array processor and made its first installation at the beginning of 1968.

The 2938, designed in collaboration with Western Geophysical, had an instruction set of about 12 instructions with data arrays as operands. Pipelining and other hardware innovations resulted in great speed improvements. Geophysical processing could, as a consequence, proceed for almost a decade without serious hardware constraints by taking advantage of normal GP computer developments to complement the array processor.

Today's array processors come in a bewildering variety of styles and capabilities. Married to a mini, one of today's better array processors will, on seismic data, outperform a GP computer and do it at less than 25 percent of the cost of the GP machine.

Seismic reflection data today, and for some time to come, consist of observations arrayed in a three-dimensional matrix. One dimension of the matrix, the one along the line of survey, is essentially unlimited but, as a practical matter, is limited to a few thousand values. The vertical (or time) dimension of the matrix is usually about 3000 data values; and the third dimension, representing the simultaneous observations for each input pulse, is in the process of growing from a present typical value of 50 to about 500 or 1000 in the next few years. For some applications in which an area is surveyed in detail (so called 3-D surveys), a fourth dimension of a few tens or hundreds of values is appended.

It is becoming more common to use extended inputs such as coded or pseudo-random pulse sequences—a process which greatly increases raw data volumes.

The only way all of these data will ever be processed is by extending the technology of specialized devices exemplified by the array processor.

I visualize the processor of 1985 built around a gigabyte high-speed memory with multiple access modes. Only a handful of algorithms will have to be applied to the data. The processor will have hundreds or perhaps thousands of identical single-chip microprocessors to execute each of the required algorithms. The micros will be used in a combination of parallel and pipeline modes so that all are operating simultaneously. All of this activity will be under control of a relatively small CPU which can handle traffic control and the calculation of processing parameters.

In operation all or a sizable chunk of the raw-data matrix will be read into main memory from the compact storage medium on which they were recorded in the field. A succession of operators will be passed through the matrix in various directions and various orders by directing sequences of data from memory through the micros.

Technology visible today can make the whole project feasible; and as Jules Verne has so aptly said, "What one man can imagine another man can do."

Seismic modeling with minicomputers

by R. L. SELZLER, R. M. ALFORD, K. R. KELLY, N. D. WHITMORE, and H. M. WILLIAMS

Amoco Production Company
Tulsa, Oklahoma

INTRODUCTION

The energy crisis has created increased interest in petroleum exploration. Discoveries of major deposits are increasingly rare and difficult to locate. The geophysicist requires more accurate knowledge to assist him in locating deeper and smaller hydrocarbon deposits. While gravity, magnetic, and electrical methods are also used, the predominant exploration tool is the seismic method and provides a wealth of subsurface information. Geophysicists continue to increase their seismic proficiency by using such things as modeling.

The concern of this paper is the ability to model the seismic response of a known subsurface. The practicality and usefulness of a modeling technique is determined by how accurately, routinely, and economically it may be used to simulate realistic cases. Modeling techniques, such as numerical solutions to the wave equation and ray tracing, are possible and offer distinct advantages and disadvantages. The method presented in this paper is a numerical solution to the wave equation obtained by finite difference techniques.

A brief background in seismic methods and theory is presented. The numerical modeling technique is described, and a typical example is given. Also discussed is a minicomputer system which makes the method economically feasible for routine use.

GEOPHYSICAL SEISMIC METHOD

The typical method of gathering seismic data is depicted in Figure 1. At the site of interest, seismic wave transducers, or seismometers, are coupled to the surface of the earth along a line typically 10,000 feet in length. Individual seismometers (actually arrays) are generally spaced about 100 feet apart and are monitored by a central recording station. A seismic source, often an explosive charge buried beneath the surface, generates seismic waves which travel through the subsurface according to the principles of elastic wave propagation. The direct wave and the many reflections from the subsurface interfaces are digitally recorded at the surface for several seconds. These records contain complex multiple reflections and ambient noise; they are later subjected to digital processing in an attempt to remove these effects.

The records are often plotted as a seismogram as shown in Figure 2. Here the abscissa represents the horizontal position of the semismometers and the ordinate represents time. Time increases downward and is equal to zero at source initiation. The contribution to the plot from a single seismometer is highlighted. It is informative to notice the reason for this orientation of the display. It is a first attempt at inferring the geological structure from the recorded time response. A sense of horizontal position is somewhat preserved while the time ordinate is related to the geological depth.

Ideally, one would like to directly generate a picture of the subsurface geology from the recorded surface data. This would be referred to as the inverse problem. Unfortunately at this time it is only possible to deduce the structure, indirectly, after an involved sequence of digital processing, followed by extensive human interpretation. Mathematical modeling can be extremely helpful during the interpretation and processing steps.

While the microscopic composition and porosity of the rocks is important, we are concerned only with their macroscopic effects upon seismic wave propagation. The earth's subsurface, for our purposes, may be divided into homogeneous regions characterized by constant velocity and density values. This defines the subsurface in terms of the parameters in the wave equation. The variation of these parameters at the interfaces between regions gives rise to reflections, refractions, and other wave phenomena.

FINITE DIFFERENCE METHOD

Given a governing equation for seismic wave propagation, it is possible to numerically approximate its solution by finite difference techniques. Although the elastic wave equation governs seismic wave propagation, the simpler acoustic wave equation provides useful and practical results and will be considered here. The same approach has been applied to the elastic equation.[1]

The two-dimensional acoustic wave equation to be considered is

$$\rho v^2 \left[\frac{\partial}{\partial x}\left(\frac{1}{\partial}\frac{\partial}{\partial x} U\right) + \frac{\partial}{\partial z}\left(\frac{1}{\rho}\frac{\partial}{\partial z} U\right)\right] = \frac{\partial^2}{\partial t^2} U, \qquad (1)$$

where $v(x,z)$ is velocity, $\rho(x,z)$ is density, U is the acoustic pressure, t is time, and x and z are horizontal and vertical

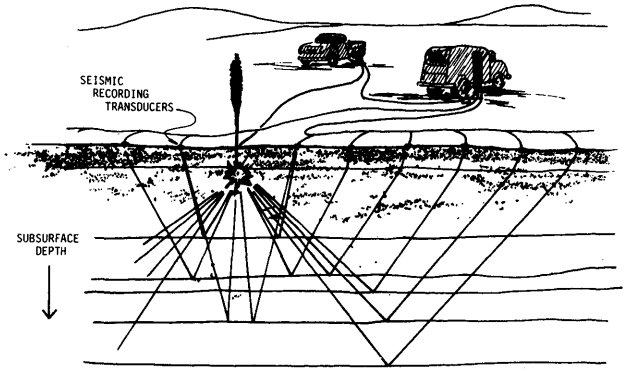


Figure 1—Seismic exploration method

position, respectively. Interfaces are automatically accounted for since velocity and density variations are included in the equation.

The majority of the calculations occur in regions with constant density, where equation (1) simplifies to

$$v^2\left[\frac{\partial^2}{\partial x^2}U+\frac{\partial^2}{\partial z^2}U\right]=\frac{\partial^2}{\partial t^2}U. \tag{2}$$

For simplicity, attention will be limited to equation (2).

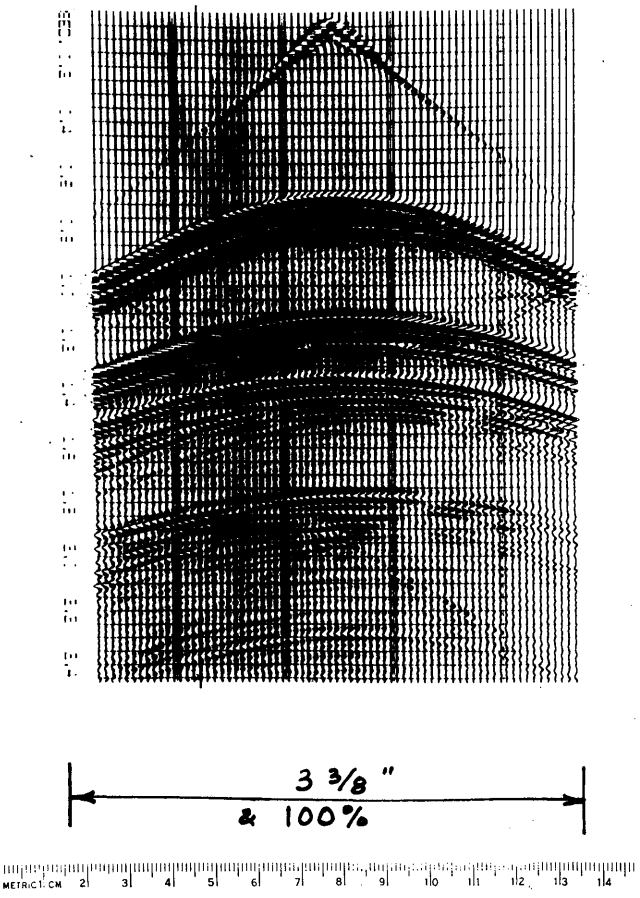


Figure 2—Synthetic seismogram

Using standard finite difference methods,[2] equation (2) may be approximated by

$$\begin{aligned}U(x,z,t+\Delta t)=&(1-2p^2)U(x,z,t)\\&+p^2\{U(x+h,z,t)+U(x-h,z,t)\\&+U(x,z+h,t)+U(x,z-h,t)\}\\&-U(x,z,t-\Delta t),\end{aligned} \tag{3}$$

where $$p(x,z)=\frac{v(x,z)\Delta t}{h},$$

where Δt is the time increment and h is the spatial grid interval.

This finite difference scheme is one of many possible approximations. Notice that it provides an explicit solution for the acoustic pressure at a future time, $t+\Delta t$, in terms of the pressure at the current time, t, the previous time, $t-\Delta t$, and the geological paramters.

The geology to be modeled is represented by two-dimensional arrays of velocity and density as shown in Figure 3 ($h=\Delta x=\Delta z$). A uniform square grid is superimposed on the model, and at each discrete grid point, values for the velocity and density are assigned. The creation of these discrete arrays describing complex geological structures is well suited for interactive graphics programs.

Consider the "molecule" of data required for the calculation of the pressure field at a single point. The velocity and density parameters do not vary as a function of time, whereas the field value does. The algorithm may be visualized as a single matrix of geological parameters superimposed on many multiple time planes of field values. Fortunately for storage conservation, the algorithm requires that only two successive time planes ever be present, since the future value can replace the previous value in storage.

The accuracy of the algorithm is a function of the coarseness of the grid. For our purposes, a minimum of ten grid points per wavelength is usually sufficient.[3] There is also an upper limit imposed upon the time step to maintain the stability of the algorithm.

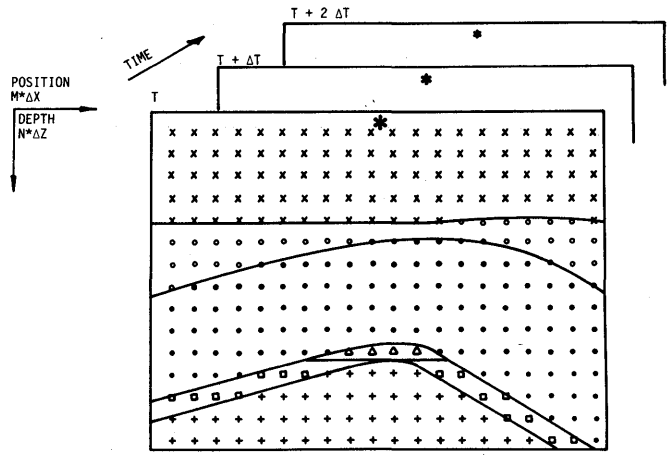


Figure 3—Geological parameters in discrete space superimposed upon multiple time planes of the wave field

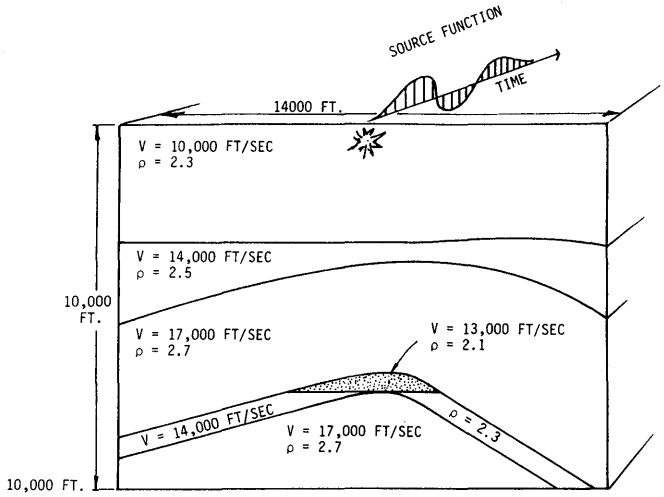


Figure 4—Geological model

MODELING APPLICATION

Figure 4 depicts the two-dimensional geological cross section of a model. The small semicircular shaped region near the center represents a gas reservoir. A synthetic seismic response is desired in order to better understand how such a deposit would be observed in the presence of the other geology.

The model extends 14,000 feet horizontally and 10,000 feet vertically, which for a grid point spacing of 25 ft, results in 560 grid points horizontally and 400 grid points vertically. The response is generated for 2500 milliseconds to assure that reflections from the depth of interest are obtained. This requires about 2000 time steps. The time variation of the source is represented by the second derivative of a Gaussian,[3] with an upper-half-power frequency of 30 Hz, which is similar to that generated by an explosive charge in the field.

Figure 2 is a plot of the synthetic time section obtained by recording the numerical values at the surface of the model. This corresponds to what the seismometers would record in the field. The first event to arrive at the surface is the direct wave from the source. Next is the reflection from the first interface, followed by many additional primary and multiple reflections. Although the geological model is relatively simple, the time section is complex and contains many refraction, reflection, and diffraction events. In practice, the responses for a large number of source locations are combined through an involved processing sequence, to create a more easily interpretable seismic section. An example of such a process, applied to a collection of synthetic seismograms for differing source locations, is shown in Figure 5. This synthetic section now more closely resembles the model geometry, and the lower boundary of the gas reservoir is readily identifiable as indicated on the display. Processing and interpretation of synthetic data sets simulating the actual

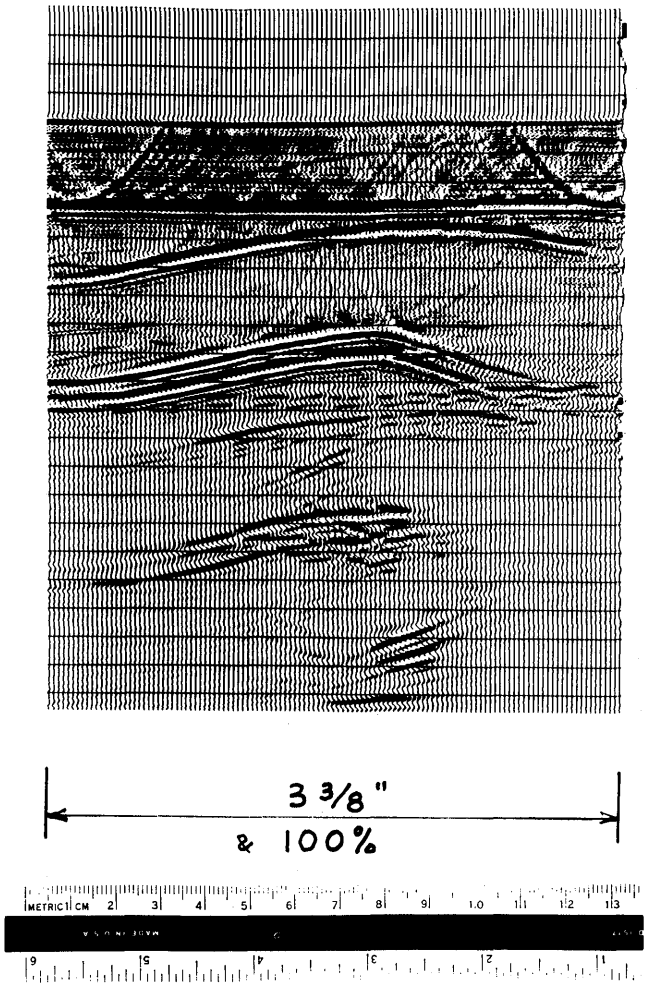


Figure 5—Composite of multiple seismograms after processing

field procedure provides an opportunity to develop and test ideas in a controlled, known, noise-free environment.

An added bonus of this modeling method is "snapshots," shown in Figure 6. A snapshot is a plot of the entire wave field at a single time plane. The geological structure has been superimposed on the display to assist the interpretation. The snapshots are resampled by four to one to conserve storage, thus causing the structure to appear coarse.

Snapshots can improve our understanding of seismic wave phenomena, since individual waves can be followed as they propagate through the media. A movie of seismic wave propagation has been produced using closely spaced snapshot sequences.

The finite differences technique is a very useful and flexible method for seismic modeling. Models of arbitrary geological complexity require no unusual considerations and seismograms may be generated routinely. The synthetic response provides realistic waveforms with accurate arrival times and amplitudes. The response includes diffraction events and, if desired, the method can be applied to the

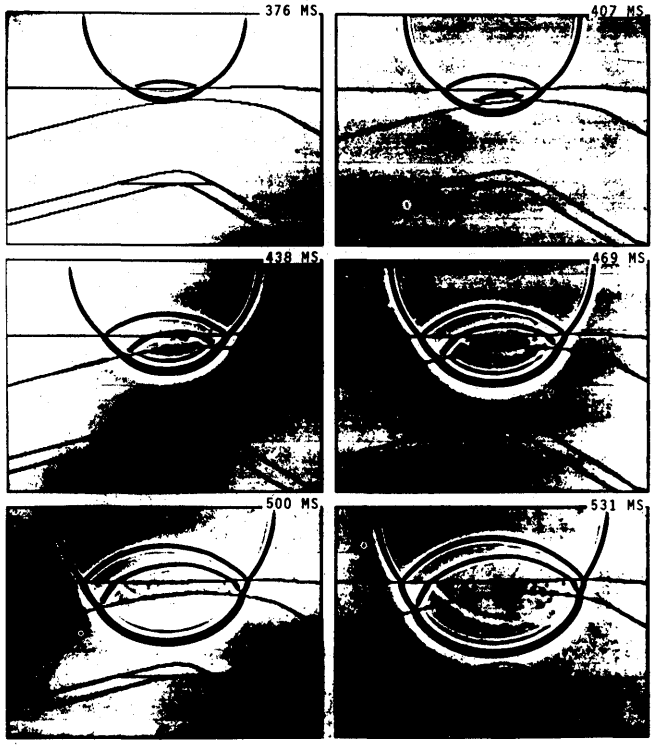


Figure 6—Snapshot sequence with structure superimposed

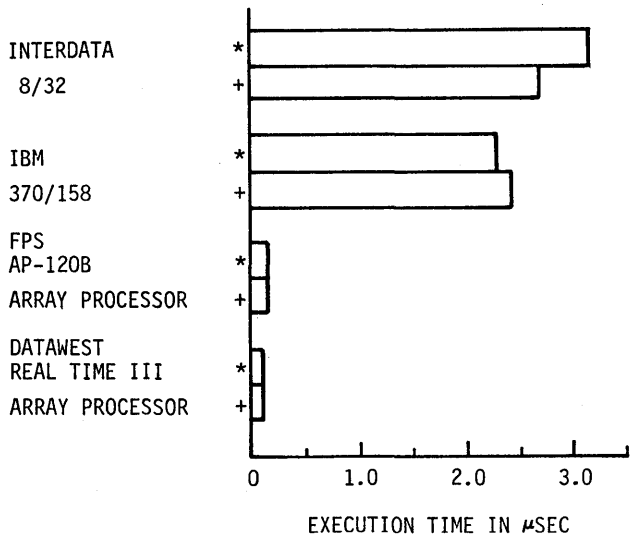


Figure 7—Multiple (*) and add (+) execution time for various floating point processors

elastic case to study shear wave events and other elastic phenomena. The flexibility of the method allows source and receiver locations to be easily varied, permitting complex arrays and non-standard acquisition techniques to be simulated.

DEDICATED MINICOMPUTER SYSTEM

The computational burden imposed by this modeling method is tremendous. Seismic models ranging in size from 400 to 800 grid points horizontally and vertically are desirable. This results in 160,000 to 640,000 grid points per model. Each geological point modeled requires from 10 to 20 bytes of computer storage. Typically the solution at 1500 to 3000 time planes must be computed. Even though a maximum of two successive time planes are needed, this requires large quantities of storage. If sufficient local memory is not available, computations may be performed on model segments rolled in and out from mass storage peripherals.

Early finite difference modeling was performed on large general purpose computers using standard FORTRAN programs. It was considered economically impractical to routinely submit models larger than 100 by 100 grid points on such a system. Minicomputers were considered as an alternative processing system.

The dominant factor in the computational speed is the large number of wave field calculations required. Calculations are performed in single precision floating point arithmetic. Figure 7 compares the floating point multiply and add speed for memory to register instructions for a variety of processors. The processors may be divided into three categories: large general purpose computers, minicomputers, and array processors. The array processors considered here are programmable devices that have their own local memory. They require a host computer and, therefore, are considered high speed computation extensions of the host.

A dedicated minicomputer system, with minimal memory and peripheral devices, may achieve real time performance comparable to a large general purpose computer for applications which are CPU bound, such as the one under consideration. In view of the comparatively modest hardware cost, the minicomputer system thus becomes economically attractive for installations where high and continued usage of the program is anticipated. The preceding modeling example required approximately eight hours of minicomputer CPU time (without an array processor) to create the synthetic seismogram for a single shot location. This would have been impractical on a large computer.

An Interdata 8/32 minicomputer was selected: it can service any subordinate function required and is capable of hosting an array processor, making available their powerful resources. The hardware configuration used is shown in Figure 8. The minicomputer has 32 bit architecture and hardware implemented floating point instructions. It supports up to one megabyte of memory with a bandwidth greater than six megabytes. Multiple selector channels allow the peripherals high speed DMA (*D*irect-*M*emory-*A*ccess) capabilities. This is important since high speed mass storage discs are an integral part of our implementation. The optional array processor also utilizes DMA extensively. A multiplexor bus is available for other low-speed peripherals.

The speed of the main calculation loop is paramount to overall performance. While the bulk of the program may be written in FORTRAN, this loop must be carefully optimized in assembly language. This is especially important with some

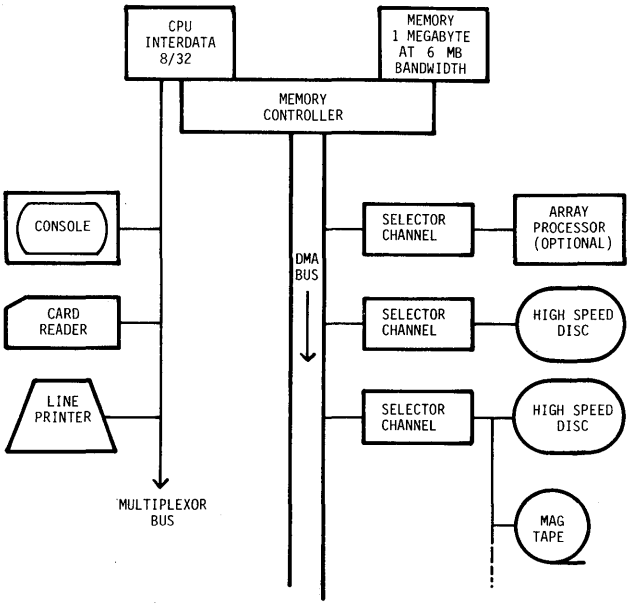


Figure 8—Minicomputer hardware configuration

minicomputer FORTRAN compilers, which may not produce very efficient execution code. This relatively simple improvement has yielded a 4-fold speed improvement. The same idea applies if an array processor is used; it is more efficient to program the algorithm directly on the array processor, than to use standard vector multiply and add sequences. Not only may direct programming improve speed, but it can also conserve expensive array processor local memory.

The continuous flow of model segments from storage, to memory, and back to storage again, is an integral part of the processing. One must be acutely aware of operating system overhead and I/O requirements for optimum performance. The hardware configuration has two high speed mass storage discs, each with an independent DMA selector channel. An unbuffered disc file access method provides the most efficient I/O. This allows input, output, and calculation of model segments to proceed concurrently. This is considerably faster and more efficient in real time than if each occurred serially. The ability to do simultaneous I/O and calculations emphasizes how the user may closely coordinate software and hardware. By carefully controlling the disc random accesses, the calculations may be synchronized to the point of transferring model segments on successive fractional rotations of a high speed disc.

CONCLUSIONS

Seismic modeling using finite difference methods provides the exploration geophysicist accurate synthetic results for complicated geological structures. Modeling provides a means of comparing synthetic results with field data. It provides a fast, flexible, and economical laboratory for investigating new procedures and confirming old ideas. The finite difference approach used here is but one of many numerical variations possible, all of which merit further consideration. This technique is not limited to geophysical applications, but is applicable to other areas of wave propagation research.

Programs demanding heavy CPU computations for long periods of time may not be justifiable on large general purpose computers. Large time shared computers are designed to obtain high usage levels through simultaneously and efficiently managing multiple tasks. Demanding programs, such as finite difference modeling, utilize the CPU for long periods of time which eliminates the need for sophisticated multiple user management. Dedicated minicomputer systems may be more economical because the memory and peripheral devices can be tailored to the specific application.

ACKNOWLEDGMENTS

We wish to thank M. Thornton, R. Read, and M. Farmer for their valuable assistance and advice on the project. Thanks are due to Amoco Production Company for permission to publish this work.

REFERENCES

1. Kelly, K. R., R. W. Ward, S. Treitel, and R. M. Alford, Synthetic Seismograms: A Finite-Difference Approach, *Geophysics*, 41, 1976, pp. 2-27.
2. Mitchell, A. R., *Computational methods in partial differential equations*, John Wiley & Sons, New York, 1969.
3. Alford, R. M., K. R. Kelly, and D. M. Boore, Accuracy of finite-difference modeling of the acoustic wave equation, *Geophysics*, 39, 1974, pp. 834-842.

Present and future applications of computer technology to petroleum exploration

by R. A. TERNUS and A. S. HOFFMAN
Chevron Oil Field Research Program
La Habra, California

INTRODUCTION

In this paper we will summarize the steps involved in exploring for oil and gas, review the role that computers play today in exploration and discuss some of the roles that computers may play in the future.

PETROLEUM EXPLORATION

The ultimate way in which oil and gas are discovered is to drill into the ground. The objective of all prior steps in the exploration process[1] is to identify where to drill to minimize the risk of drilling "dry" holes.

Generally exploration (Figure 1) begins with a regional geologic survey. This is done to locate geologic basins containing sedimentary rocks, and to extract information from surface geologic features and rock samples about the types of sedimentary rocks and their depositional environments. Such information can suggest where oil and gas may have been generated and is trapped in reservoirs.

The second step may consist of gravity and magnetic surveys. These determine the regional variations in the thickness of the sedimentary rocks and locate subsurface structures such as faults and salt domes, which often help to form traps for hydrocarbons.

Gravity prospecting involves measuring local fluctuations in the earth's gravity field, then correcting these for latitude, elevation, topography, and earth tides. The resulting gravity maps can be useful in determining subsurface geological features marked by changing rock density.

Magnetic prospecting, which measures local fluctuations in the earth's field, is more complex and erratic than gravity measurements. However, the magnetic measurements are simpler and cheaper to make, require only minimal corrections, and are more diagnostic of mineral distribution.

The third step consists of reflection seismic surveys, some of which may be long regional lines used to better define the sedimentary basin. Others are laid out on more finely-spaced grids—to obtain detailed information in specific areas. From these one can determine the specific locations of possible hydrocarbon traps and, in some cases, the actual presence of hydrocarbons.

The basic principles (Figure 2) of the seismic reflection method[2] are rather straightforward. A controlled source of

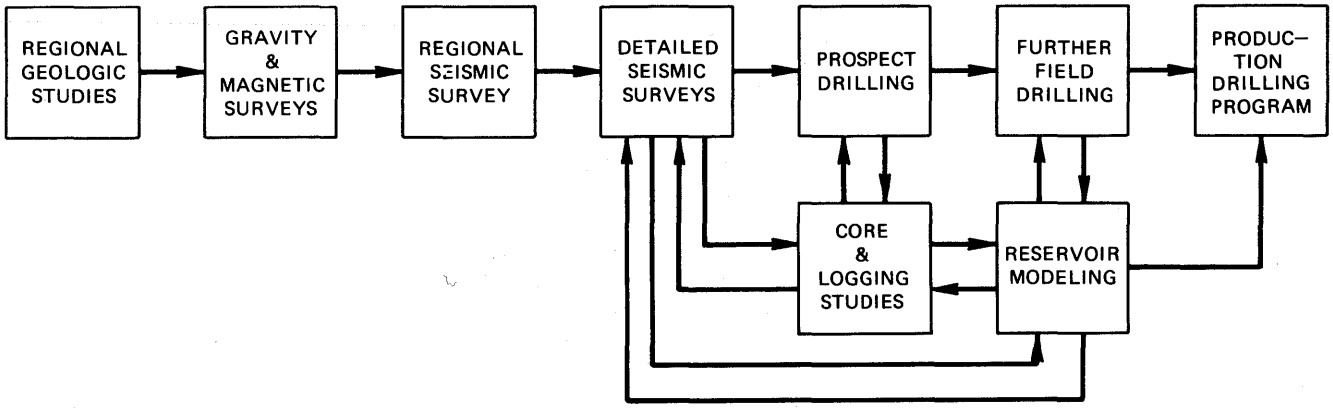


Figure 1—The exploration process

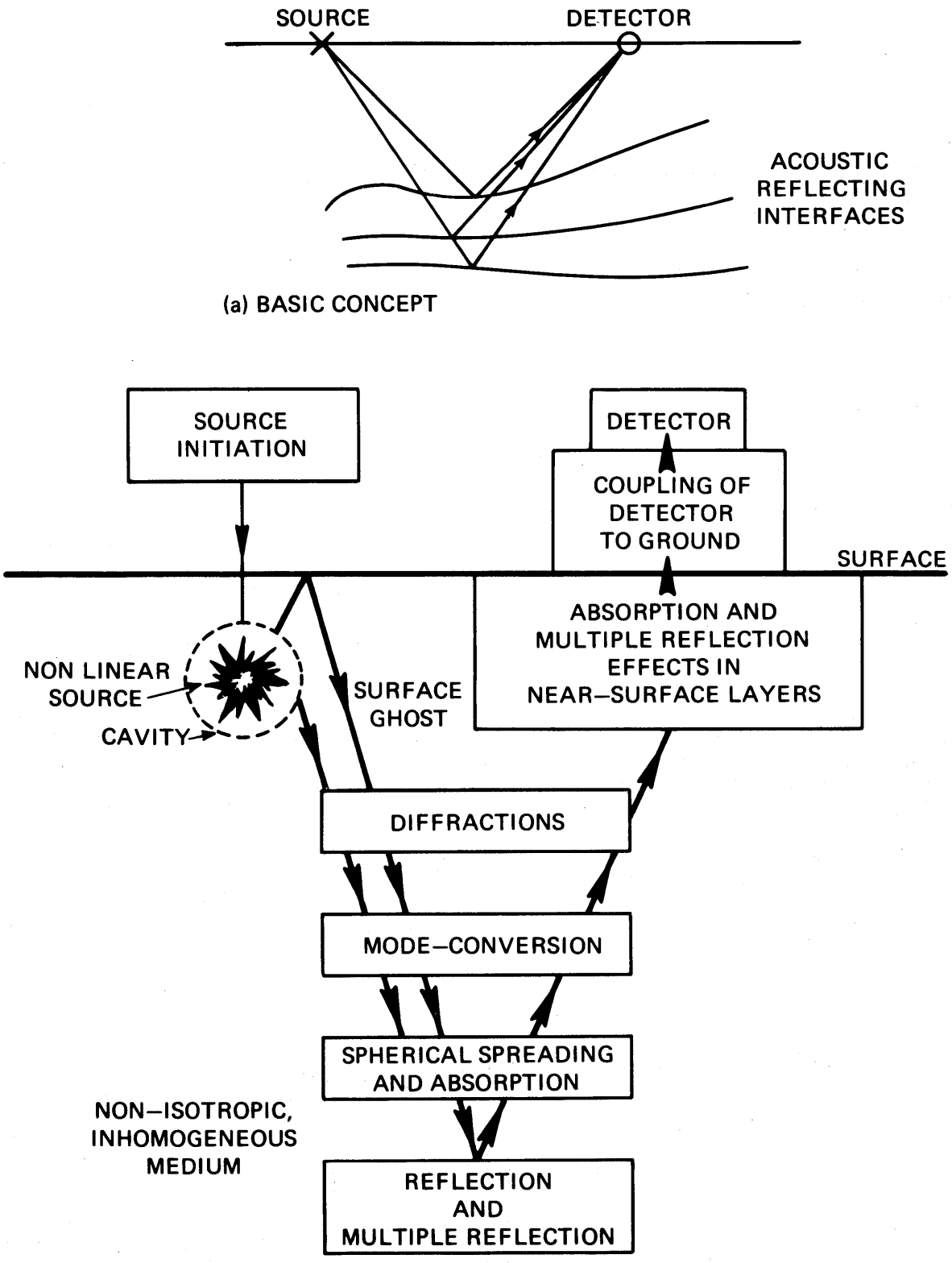


Figure 2—Illustration of the seismic reflection method

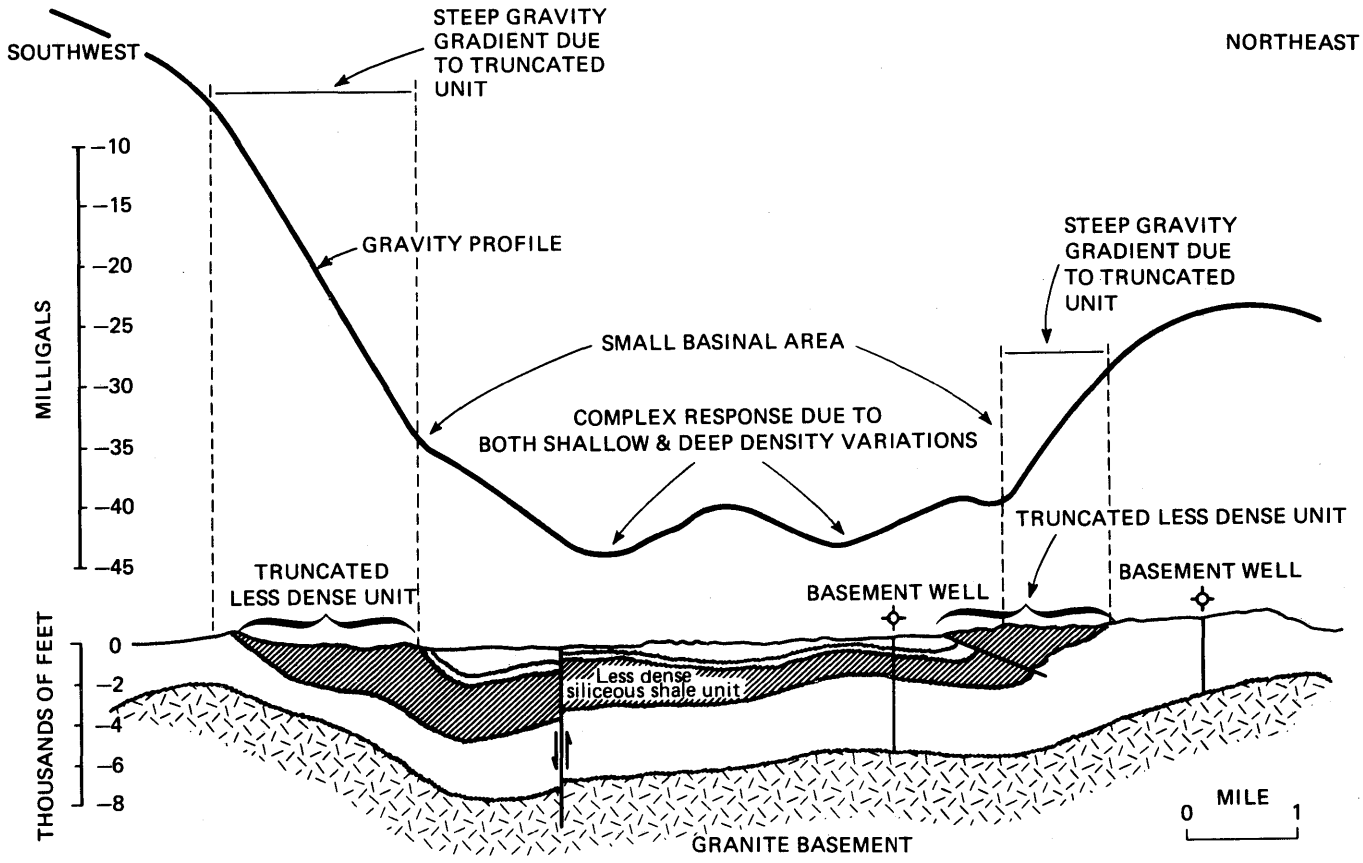


Figure 3—Gravity profile and geologic section

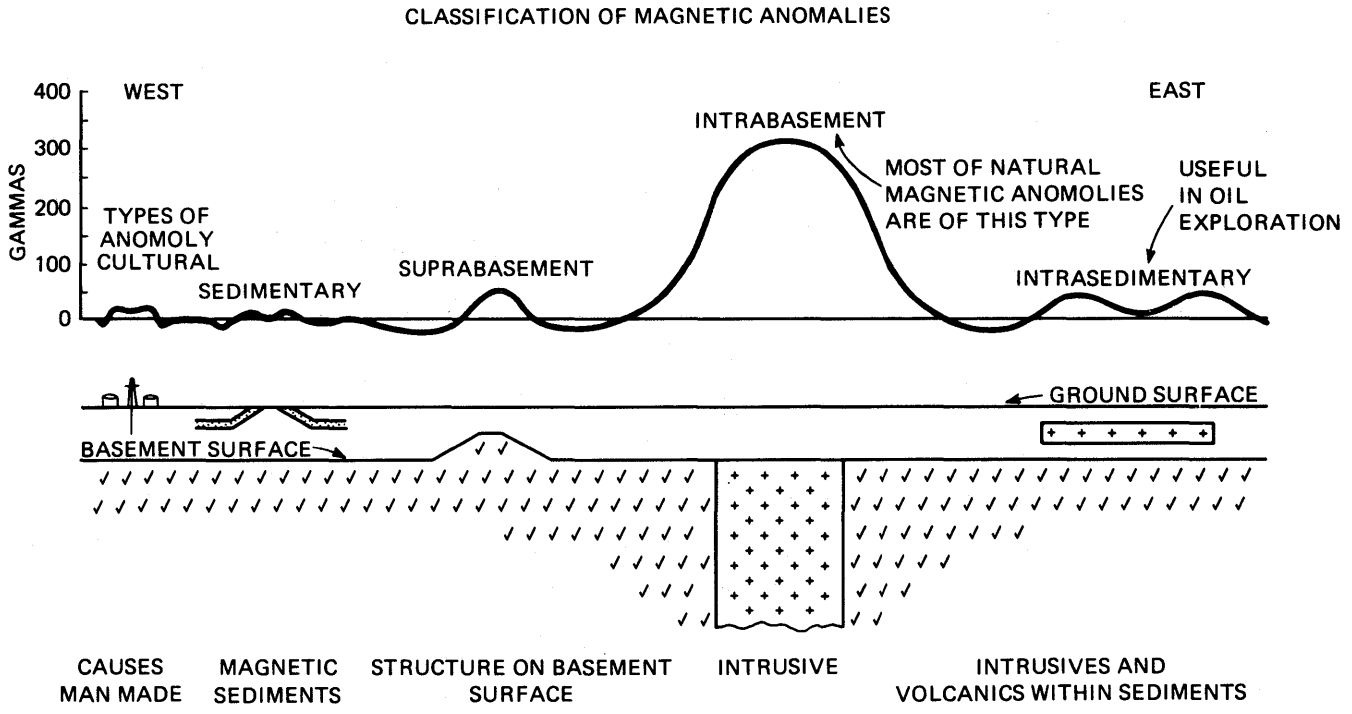
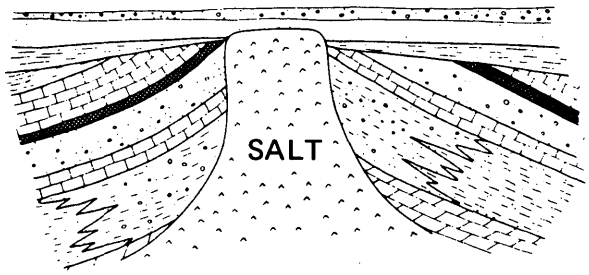
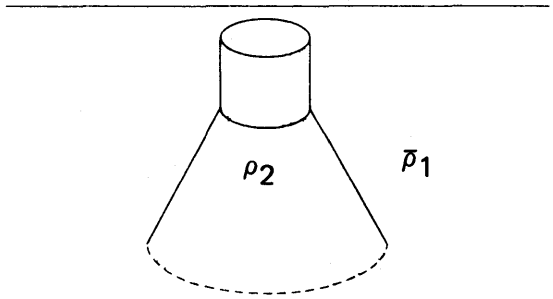


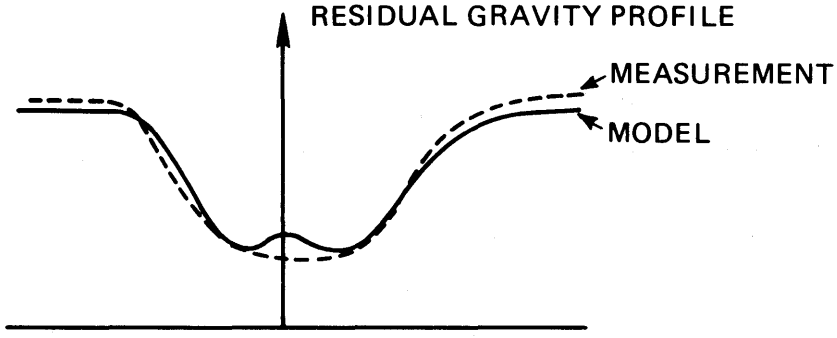
Figure 4—Magnetic profile and geologic section



(a) ACTUAL GEOLOGIC STRUCTURE



(b) DENSITY MODEL OF STRUCTURE



(c) CORRESPONDING GRAVITY PROFILE

Figure 5–A salt dome and its interpretation model

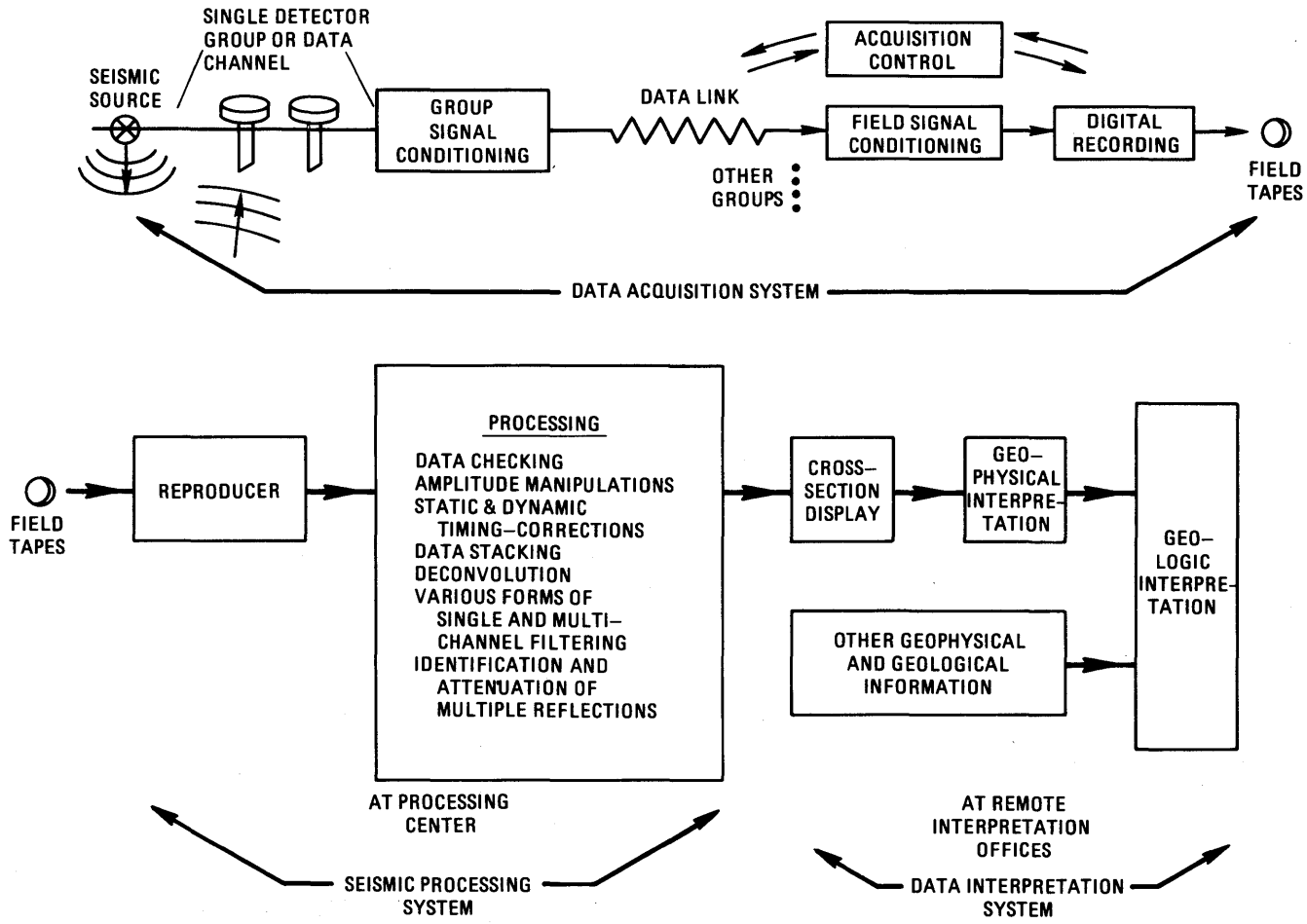


Figure 6—Seismic reflection data flow

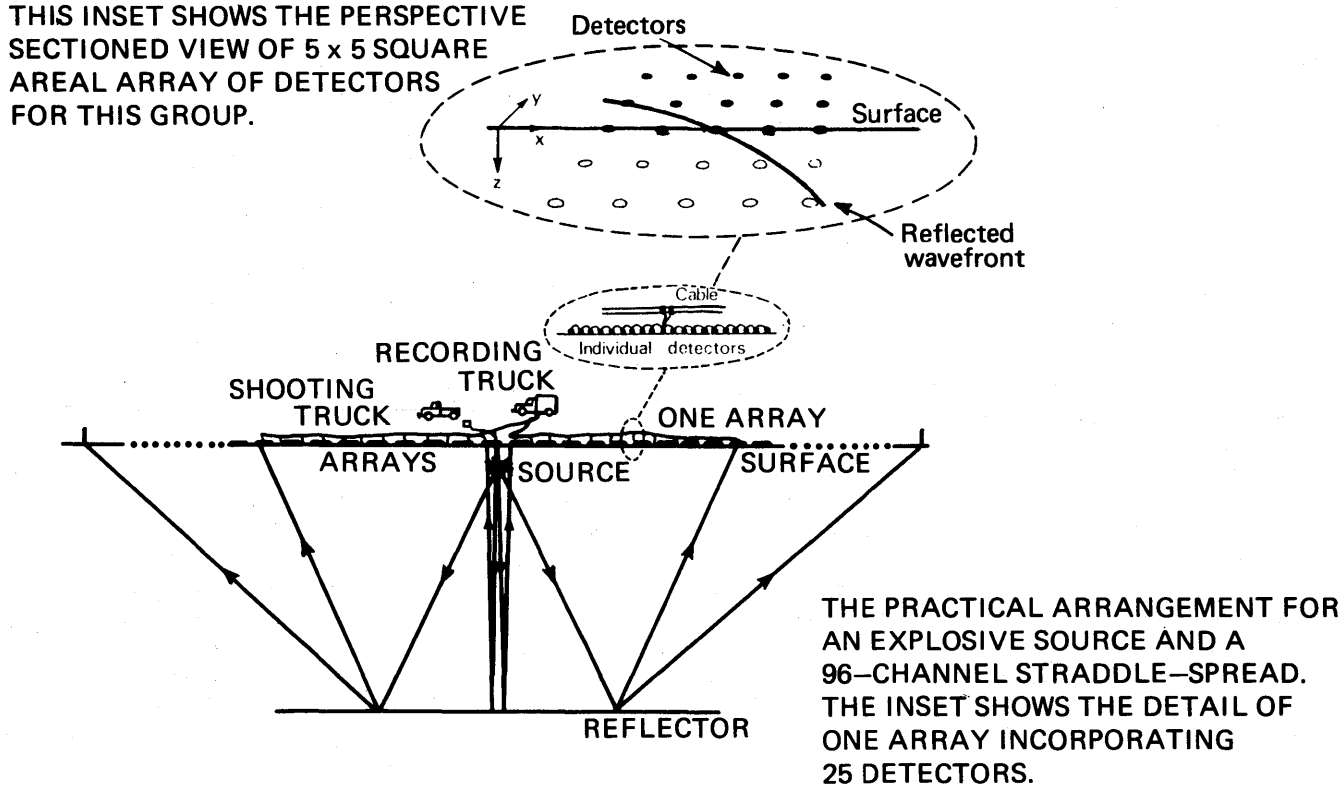


Figure 7—Seismic data acquisition spread

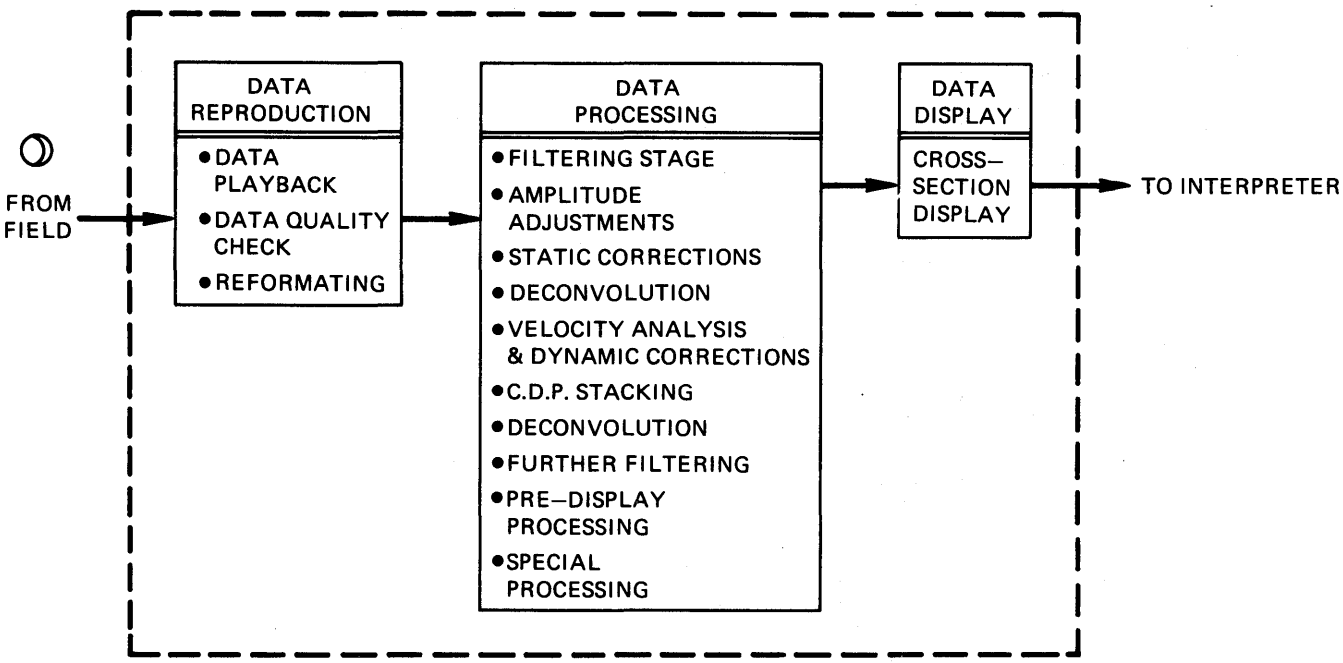


Figure 8—Typical processing flow

acoustic energy at or near the earth's surface propagates energy into the subsurface. The reflected energy from geologic strata is sensed by arrays of seismic detectors at the earth's surface (or in the sea, slightly below the surface).

Correlating the resulting subsurface reflectivity maps with a knowledge of geologic structure and rock properties allows the interpreter to delineate a three-dimensional geologic model of the subsurface. The basic acquisition objective of the seismic technique is to maximize the contribution from compressional waves reflected from the depths of interest and to minimize that from surface waves and ambient noise.

CURRENT USE OF COMPUTERS

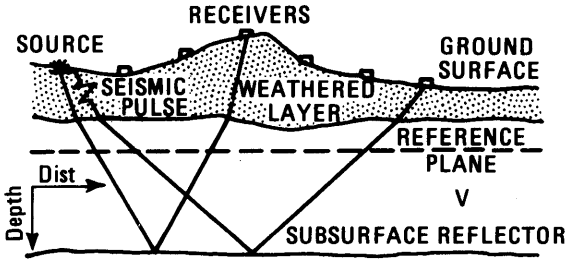
Computers are used extensively today in all phases of exploration, with the possible exception of the initial regional geologic surveys. They are used to filter the gravity and magnetic data, make various geometric-type corrections, and assist in interpreting the data (Figures 3 and 4). The latter is often accomplished by constructing mathematical models that describe the shapes, densities, and/or magnetic susceptibilities of various parts of the subsurface. The implied gravity and magnetic fields for these models are calculated and compared to measured fields. The mathematical models are then adjusted to obtain suitable matches between implied and actual fields. The final model (Figure 5) provides an interpretation of the data.

However, the most widespread use of computers in exploration is in the seismic reflection method. There is a ubiquitous presence of various types of data-processing systems, including data acquisition in the field, "number-crunching" data processing and reduction, and the often-interactive interpretive phase (Figure 6).

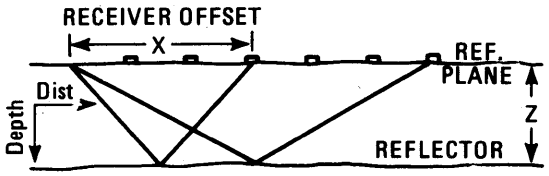
The data acquisition stage (Figure 7) involves several thousand individual detectors connected in 100 or so separate recording channels. Each channel is sampled at up to 1000 times per second, usually over the 4-6 seconds of useful reflection time for a single seismic source event. Depending on whether the acoustic source is below or on the surface, from about a hundred to several thousand such events can be initiated each day and either recorded or partly "stacked" in the field and then recorded.

The objective in the processing stage[3,4] is to enhance the desired signal at the expense of the noise, to make geometric corrections to the data as well as to extract desired parameter values. A typical processing flow (Figure 8) might be as follows: magnetic tapes containing the field data are played back in the processing center, and the data is sorted and properly formatted for processing. Traces, having either excessive noise or neither noise nor data (i.e., dead traces) are edited out or flagged so that they do not interfere with future processing.

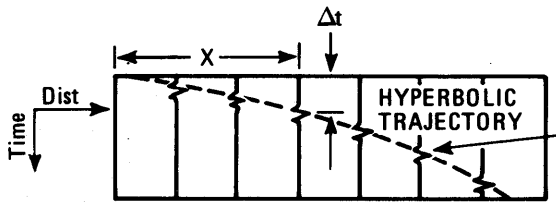
Next, the data may be filtered to remove noisy parts of the frequency spectrum that contain little or no signal. Static adjustments (time shifts) are calculated and applied to each



(a) DEPTH MODEL WITH LOW–VELOCITY LAYER OR WEATHERED LAYER



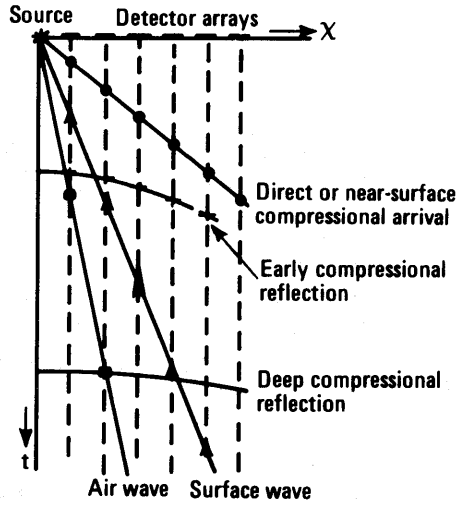
(b) DEPTH MODEL AFTER STATIC CORRECTIONS



$$\frac{\Delta t}{t_0} \approx \frac{X^2}{2V^2}$$

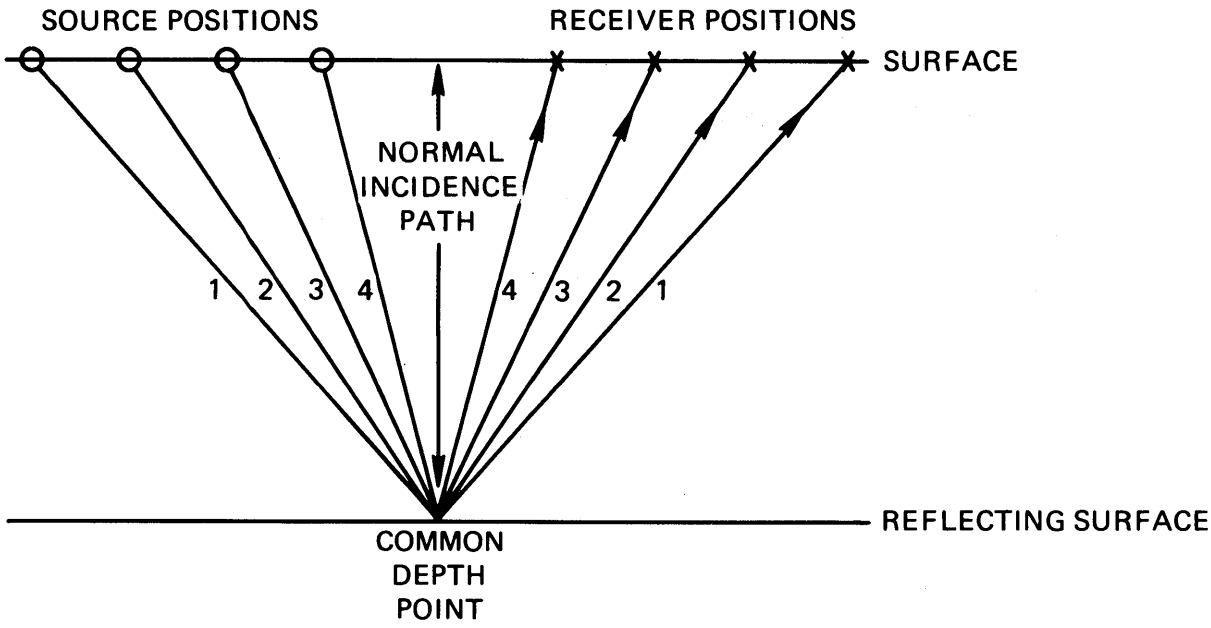
(c) STATIC CORRECTED REFLECTION RECORD

WHERE V IS SEISMIC VELOCITY AND t_0 IS VERTICAL TRAVEL TIME TO REFLECTOR



(d) VARIOUS TYPES OF SEISMIC WAVES SEEN ON A RECORD

Figure 9—Illustration of static and dynamic corrections



COMMON DEPTH POINT (CDP) STACKING
SUMMATION OR STACKING OF TRACES SHOWN (REFLECTING FROM A COMMON DEPTH POINT) PRODUCE A SINGLE COMPOSITE TRACE. PROCESS IS SUCCESSFUL BECAUSE PRIMARY REFLECTIONS FROM CDP ADD IN PHASE WHILE AMBIENT NOISE AND OTHER SEISMIC EVENTS ADD OUT OF PHASE.

Figure 10—Common depth point stacking

trace, to compensate for variations in shot and receiver elevation and in the thicknesses of shallow, low-velocity sediments. The velocities in the subsurface are estimated by examining the different arrival times of reflections associated with different shot-receiver offset distances (Figuare 9) but the same reflection point in the subsurface. Deconvolution—a filtering type of operation that serves to "whiten" the remaining part of the spectrum—is normally applied. This improves the resolution between closely spaced events. Finally, all traces having the same common depth point (Figure 10) may be summed to create a final stacked section. At various points in this processing stream, special techniques may be employed to enhance the coherent signals and to reduce multiple reflections.

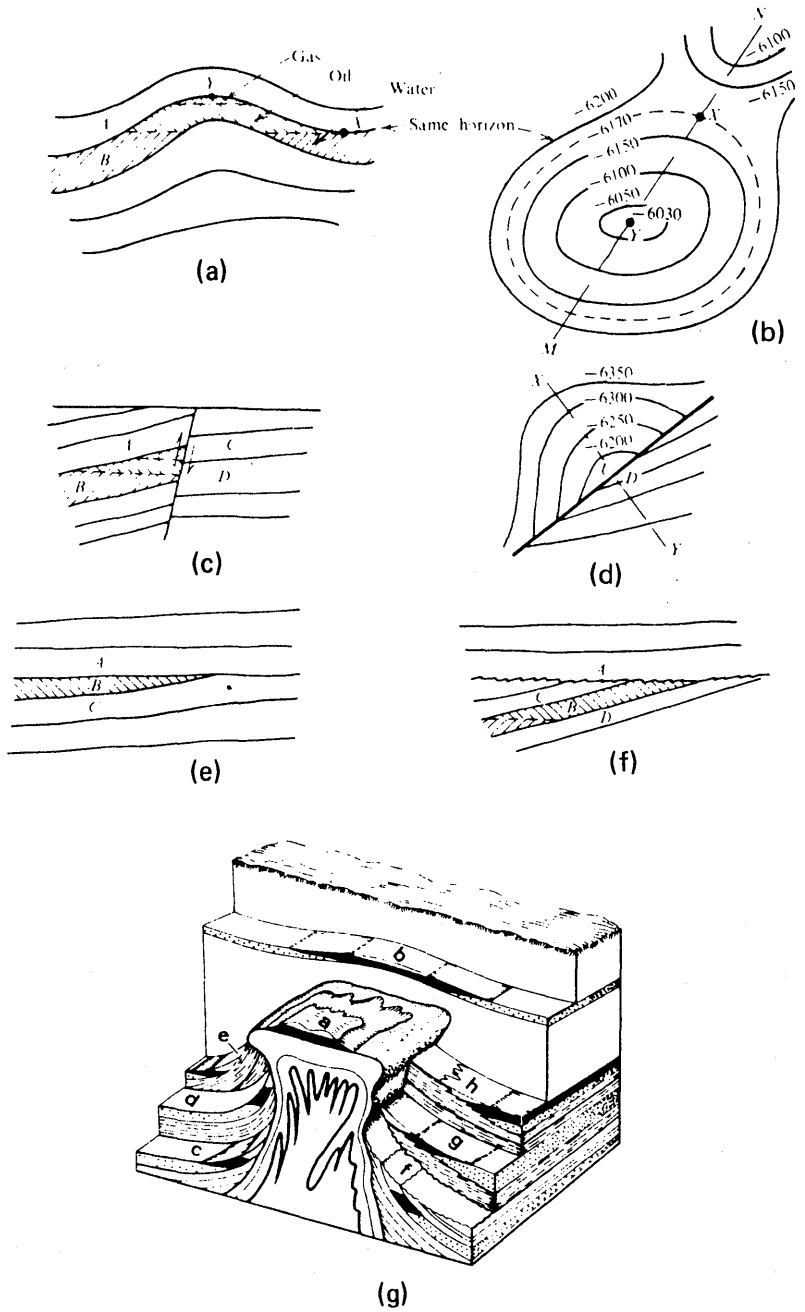
Computers are also used to interpret seismic data in a manner analogous to that described previously for gravity and magnetics data.

As the data is collected and interpreted, maps and seismic cross sections are constructed (Figures 11 and 12) showing the configurations of the sedimentary rock layers (beds) and possible locations for trapped hydrocarbons. Interactive graphics terminals are often used so the interpreter can work with the computer in the construction of these maps.

After these initial steps a well (or wells) may be drilled. Whether or not hydrocarbons are found in the drilling, data about the rocks and their pore fluids are incorporated with the previously obtained data, to refine the interpretations and maps and identify additional drilling locations.

FUTURE USE OF COMPUTERS

In the future the use of computers will continue to expand in the roles of processing and interpretation. Processing algorithms often are built around simplifying approximations, made to hold down computer processing time. As



SEDIMENTARY STRUCTURES WHICH PRODUCE OIL TRAPS. (a) VERTICAL SECTION THROUGH ANTICLINE ALONG LINE MN IN (b); (b) MAP OF HORIZON A IN (a); (c) VERTICAL SECTION THROUGH FAULT TRAP ALONG LINE XY IN (d); (d) MAP ON TOP OF BED B IN (c); (e) PINCHOUT; (f) UNCONFORMITY TRAP;

(g) OIL TRAPS ABOVE AND AROUND A SALT DOME. a) CAPROCK FIELD OF ANTICLINAL TYPE WITHIN THE TOP OF THE SALT DOME. b) FIELD OF ANTICLINAL TYPE IN LAYERS ABOVE THE DOME. c) FAULT TYPE FIELD AT THE FLANK OF THE SALT DOME. d) FAULT TYPE FIELD BELOW THE OVERHANGING TOP. e) FAULT TYPE FIELD BETWEEN TWO RADIAL FAULTS. f) FIELD BELOW A DISCONFORMITY, WHICH HAS BEEN CAUSED BY MOVEMENTS OF THE SALT. g) OVERLAP FIELD ABOVE SUCH A DISCONFORMITY. h) SHALING OUT FIELDS ON ONE SIDE OF THE SALT DOME.

Figure 11—Typical exploration prospects

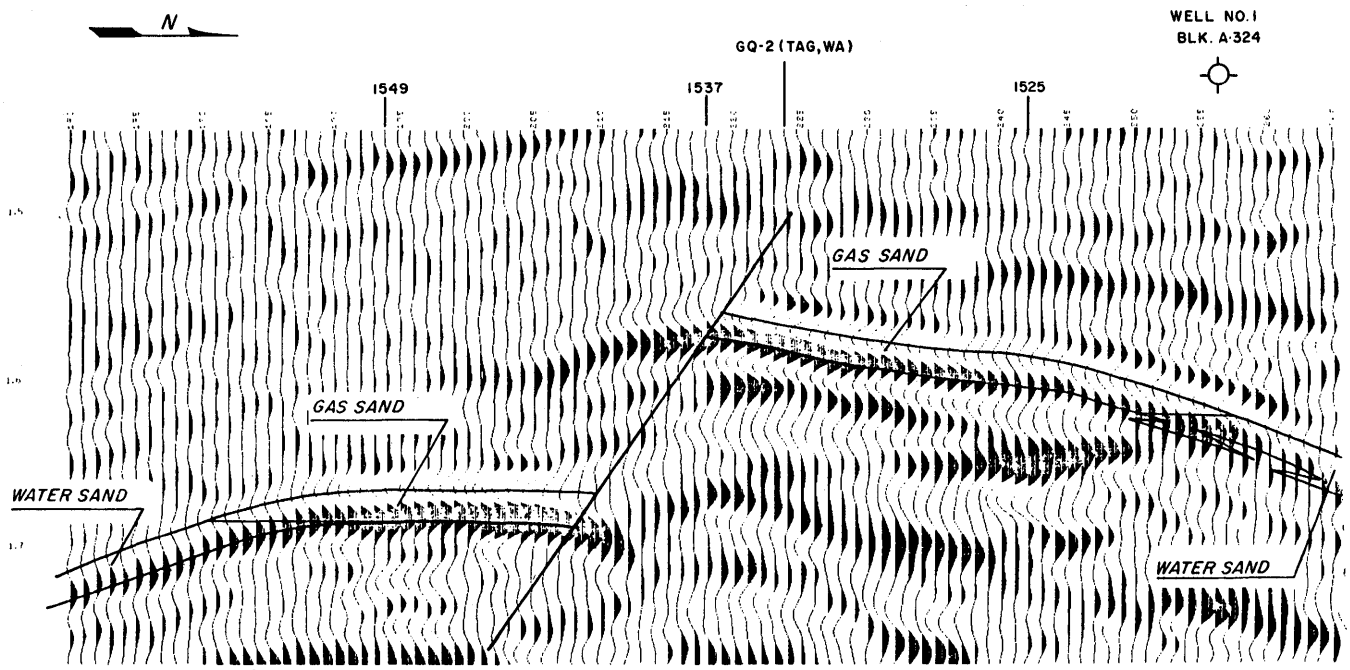


Figure 12—Seismic section showing possible oil traps

computers become more powerful, some of the approximations can be eliminated, so we can improve our processing capabilities and thereby extract useful information from data which now yield little or none. In addition, we will be more finely sampling the seismic data. This will improve resolution, accuracy, and sensitivity in our interpretations of the subsurface by increasing the bandwidth of the data.

While much of this processing is done on large general-purpose computers, we have, over the last few years, begun to make more use of special-purpose computers, such as programmable array processors, and this trend is certain to continue. The use of interactive graphics is still in its infancy, and we will see considerable growth in the use of this type of computer aid. Consider, for example, the problem of reordering seismic data. Some of the processes mentioned above operate on one trace at a time. Others need all of the traces originating from a single shot. Still others need all of the traces having a common depth point; that is, those having the same midpoint between their respective shots and receivers. As we work with more traces, because of the need for higher redundancy, we will need a "black box" that can rapidly reorder the data prior to the individual processing steps.

Preprocessing during data collection will also require more special-purpose systems. Consider the current limitations of some of our data collection procedures. In a normal seismic survey, multiple (10-100) sensors, i.e., geophones or hydrophones, are connected to collect the data for a single trace. This increases the signal-to-noise ratio at the output of the sensor group over that available from the individual sensors and reduces (by spatial filtering) the effect of unwanted energy propagating along the surface. However, this type of configuration has limitations, particularly in recording higher frequency seismic data. In the last few years, we have been going in the direction of recording more groups, so that the length of each can be reduced.

Ultimately we may record each sensor individually. However, this leads to the collection of large volumes of data that must be transported back to the processing center and put through on the computer. This method may be limited to areas requiring the highest data quality obtainable.

In the next five years one can project data acquisition rates up to 40 MBits/sec, requiring a storage capacity per record (i.e., a single seismic event for a deep source, or twenty or so coherent events for a surface source) of 350-500 MBits. For normal crew operations, this would require a storage capacity of from 10-350 GBits/day's operation. Obviously we will need either a mass storage mechanism in the field, or more processing before recording, or possibly both.

In order to obtain many of the benefits of recording each sensor individually, we will probably go in the direction of preprocessing at the group level in the field (Figure 13) rather than in the processing center. The rapid growth of the computer hardware technology is well-known, particularly the increasing capabilities of micro-processors and the increased density of memory chips. It is reasonable to expect that within the next few years we will develop the hardware and software algorithms necessary to perform differential dynamic corrections, differential static corrections, and

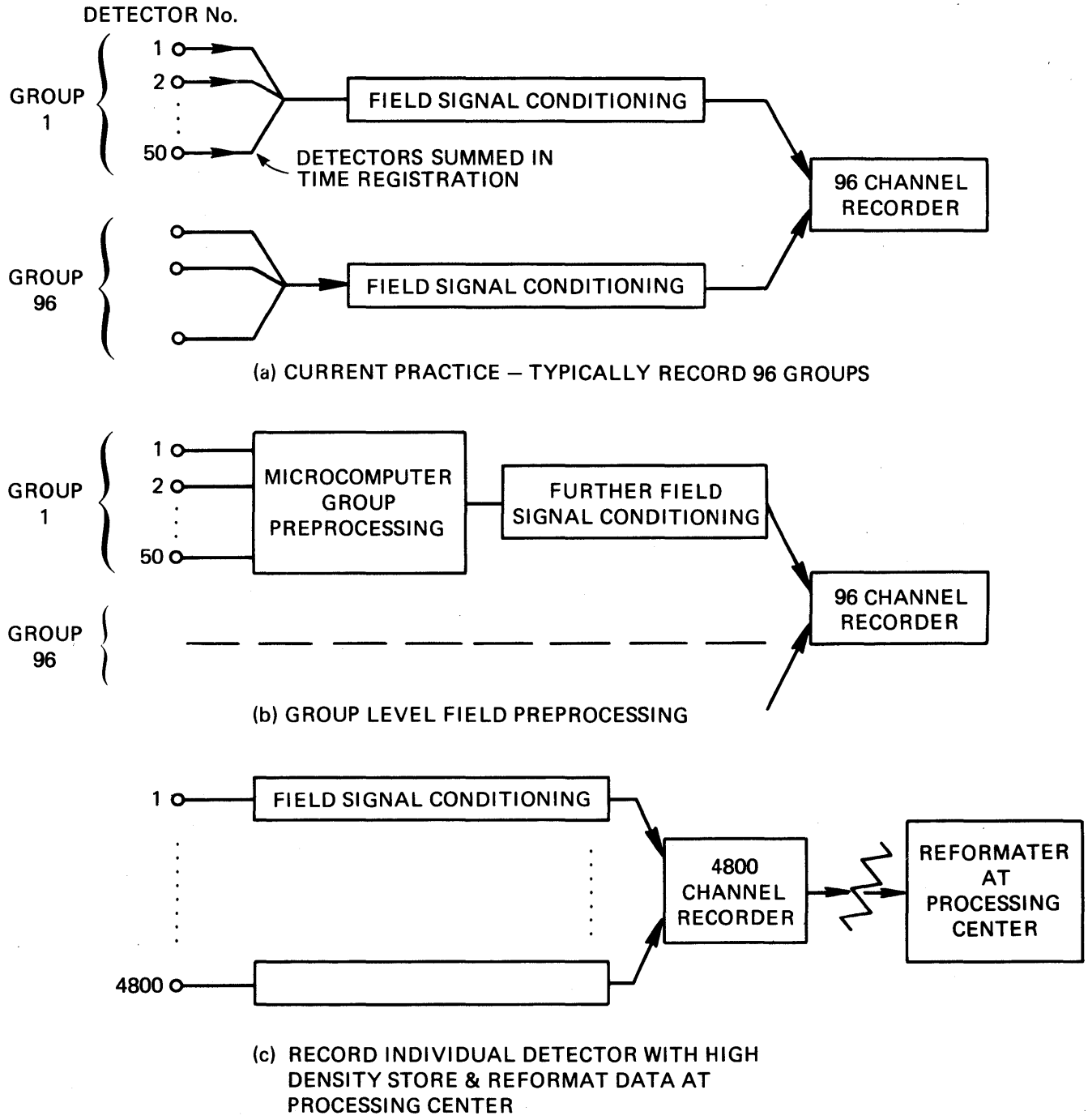


Figure 13—Group level preprocessing

waveform variation adjustments on the output signals from the individual sensors. We can then sum these signals in the field to obtain the equivalent of today's sensor group output.

In summary, computers are used extensively to assist in the exploration for oil and gas. Future processing and interpretation requirements will put greater demands on our use of computer technology, and we will have to use innovative approaches in order to solve these problems.

REFERENCES

1. Beckmann, H., *Geological Prospecting of Petroleum,* Ferdinand Enke, Stuttgart, 1976.
2. Anstey, N. A. *Seismic Prospecting Instruments,* Volume I, Gebrüder Borntraeger, Berlin, 1970.
3. Lainiotis, D. G., "Geophysical Data Processing: A Brief Survey," *IEEE Trans. Geoscience Elecs. GE-15, No. 1,* January 1977.
4. Wood, L. C. and S. Treiter, "Seismic Signal Processing," *Proc. IEEE 63, No. 4,* April 1975.

Area Director:
Frederic C. Billingsley
NASA Headquarters
Washington, DC

Remote sensing and image processing in earth resources

THE EARTH RESOURCES SITUATION

Population growth and demands for higher standards of living are creating pressures which affect all aspects of our society. The global population, currently near 3900 million, is growing at a rate of 75-80 million/year.

Coupled with population growth, and magnifying the effect, are rising demands for a more equitable sharing of the products and benefits of technology. The products and benefits derive from the consumption of natural resources. As societies develop and technological advancement begins, individual demands increase resulting in a more rapid rate of consumption of resources. Information is necessary to enable prudent decisions regarding the management of resources. But understanding alone will not lead to solutions on a long-term basis. Knowledge must be current, necessitating continuing inputs of fresh information. The most practical and economical manner of acquiring repetitive global, Earth Resources information is by means of a satellite-based remote sensing system.

Key to the utilization of the data is the ability to register the data to the ground, to previous images, and to data derived from other sources, and then to perform analysis. These together comprise the area of image processing.

IMAGE PROCESSING

The area as a whole will be covered by four sessions. They will be planned to lead persons not knowledgeable in image processing into the subject, with primary emphasis on image processing for remote sensing of the Earth. This area of application is selected because of its growing importance in the maintenance of our quality of life and the growing need for monitoring of the Earth's resources. The four sessions taken together will have some aspects of a short-short course, in that they will briefly introduce the remote sensing problem and the characteristics of images (Session I), discuss image-specialized computer techniques and programming considerations (Session II), then considerations of image processing system design and digital image based information systems (Session III). Session

IV is planned to be a panel, with presentations of future satellite and data plans and data characteristics. This is planned as a panel to allow rebuttals and discussions of these future plans and their implications to the computer world.

This format has been selected to provide cohesive treatment of the subject, with the thought that many computer people who have not been involved in it will become involved in the future as the popularity of digital image processing and analysis grows. This growth is expected based on the widespread use of the digital data to date, far beyond the early NASA expectations, and the NASA plans to provide much improved digital data in the future.

Applications have been discussed in other contexts extensively. The discussions here will lay the groundwork, but then concentrate on the implications to the computer world. These implications will include data timeliness, effects of various processes such as the necessary geometric registration on the utility of the data as seen during the subsequent analyses, and the user needs for interaction and data display.

In the applications world, images are getting larger, for example, up to 6000×6000 multiband elements for future Landsat. Data storage, dissemination, and processing are all going to be a problem, especially to the small or medium user. At the same time, the analyses being attempted are becoming more sophisticated. There is a need for image descriptive and image processing languages which will provide improved tools for the analyst. They must be flexible, efficient, and easy to use. New data dependent adaptive techniques are being devised.

The required image analysis cannot exist in a vacuum. Efficient systems must be provided. Three types are needed: (1) Large pipe-line systems for handling large quantities of data routinely and quickly; (2) Flexible, modular systems for research and/or interactive analysis; (3) Geographically oriented systems for use with/as geocoded data bases. These all have features in common: need for efficiency, need to retrieve and handle large multiple data sets (the images), possible need for interactive analysis and refreshed image displays. The latter two also have the need to be easily modularly expanded and updated. The geocoded systems must be able to input and output data in tabular and polygon form as well as imagery, and be able to use all together.

Information requirements for natural resource inventories

by WILLIAM J. BONNER
Bureau of Land Management
Denver, Colorado

INTRODUCTION

Background

The Bureau of Land Management (BLM) administers the natural resources on 191 million hectares of public lands in eleven* contiguous western states and Alaska. In addition BLM has management responsibility for 25 million hectares of publicly owned mineral resources on privately owned lands. Added to this is the management responsibility on approximately 500 million hectares of Outer Counter Shelf (OCS) lands.

The basic task of BLM is to manage all resources found on these lands to provide maximum public benefit and full consideration for good conservation practices. This includes the dedication to carrying out whatever programs are required to insure that the stewardship of the public lands and their resources leads to the optimum planned use for the long range public good. These programs include activities in domestic livestock grazing, fish and wildlife ecology and habitat development, outdoor recreation, timber production, watershed protection, wilderness preservation, minerals development, environment protection, river basin planning, and general land use classification under the concept of multiple use. Resource management and development activities are supported by various programs which provide for roads, trails, and physical improvements such as recreational facilities; and by an active program to protect the public lands and their resources from wild fires and from all forms of public and private misuse.

To carry out these programs BLM must collect and handle large quantities of resource inventory information. Further the fiscal and manpower resources which must be committed to gathering this information is staggering. This is due to (1) the increasing public awareness of the importance of wildlands as a national resource, and (2) the increasing need for documentation of the bases of management actions, including baseline resource conditions, trends, plans, and environmental analysis of proposed actions. For example, a large fraction of the vital energy resources (coal, oil, oil shale) of the nation underlie the public lands; the development of these resources in a manner consistent with national interests requires timely, accurate, and comprehensive resource inventory information.

In October 1976, the 94th Congress passed Public Law 94-579. The purpose of this Act is to establish public land policy; to establish guidelines for its administration; and to provide for the management protection, and enhancement of the public lands. In addition this Act which is known as the "Federal Land Policy and Management Act of 1976" charges the Secretary of the Interior with the following responsibility.*

> "The Secretary shall prepare and maintain on a continuing basis an inventory of all public lands and their resource and other values (including but not limited to, outdoor recreation and scenic values), giving priority to areas of critical environmental concern. This inventory shall be kept current so as to reflect changes in conditions and to identify new and emerging resource and other values. The preparation and maintenance of such inventory or the identification of such areas shall not of itself, change or prevent change of the management or use of public lands."

Thus Public Law 94-579 coupled with existing requirements causes BLM to reassess its traditional methodologies for the collection of resource inventory information.

Remote sensing in resource management

BLM is involved in a program with the National Aeronautics and Space Administration (NASA) and EROS Data Center (EDC) to evaluate remote sensing technology. The purpose of this evaluation is to determine the feasibility of performing resource inventory via remote sensing on an operational basis.

Management of the public lands under the concept of multiple use requires consideration of several resources. The BLM system identifies these resources as: range, forestry, wildlife, recreation, watershed, minerals, and lands. The common denominator of these resources is vegetation

* Arizona, California, Colorado, Idaho, Montana, Nevada, New Mexico, Oregon, Utah, Washington and Wyoming

* Public Law 94-579, Title II Section 201(a), 43 USC 1711, October 21, 1976

and for that reason BLM has selected the broad area of Wildland Vegetation as the theme for this program. The overall program objective is to test and implement a remote sensing based inventory system for wildland vegetation resources on an operational basis within BLM. It is planned to operationally implement the techniques developed during this project.

Agency roles

The Wildland Vegetation Resource Inventory Project is a joint effort between the three previously mentioned organizations. The BLM has overall management authority in the program and works closely with NASA and EROS to insure that project schedules and milestones are met and that program activities are successfully accomplished. In addition to this management function the BLM assumes the following responsibilities:

(1) Coordinate and document user activities within BLM.
(2) Perform cost analysis studies.
(3) Acquire large scale photography.
(4) Specify and evaluate output products.
(5) Implement training programs in remote sensing for BLM personnel.

NASA, working in cooperation with BLM, is providing the technical capability for the initial year of the program through a contract with a commercial remote sensing firm. As part of this contract NASA is obtaining data analysis, documentation, and specialized training for BLM personnel. In addition NASA is providing small and medium scale photography to be used as supporting data in the digital analysis processes.

EROS through EDC provides specialized facilities and equipment as well as technical support for key project activities. In addition EDC is collaborating with BLM on development and implementation of a BLM wide remote sensing training program.

The overall concept of NASA and EROS involvement is to transfer remote sensing technology to BLM. To achieve this goal a plan has been developed in which BLM rapidly assimilates the technology in house and assumes prime responsibility for performance of the project tasks. At the end of the program BLM will have an operational program in remote sensing and be independently capable of performing all activities required for wildland vegetation resource inventory.

TEST SITES

In selecting test sites for this program a number of factors were considered. It was desired to have test sites with a variety of vegetation. In addition test sites were to be representative of the public lands which BLM administers. The size of the test site was scoped at approximately one million hectares; a size judged sufficient to constitute a legitimate test of the technology but not so large as to require more than one project year to complete. The final criterion for test site selection was the presence of timber in sufficient volume to conduct a meaningful test of multistage sampling technology. Using these guidelines the BLM selected test sites in Alaska, Arizona and Idaho.

Alaska test site

The Alaska test site is located in the interior of Alaska and encompasses approximately one million hectares. The northern boundary of the site is roughly the foothills of the Alaska range. The southern boundary is defined by an east-west line passing through the confluence of the Maclaren and Susitna Rivers. The test site extends from the town of Paxson on the east to Cantwell on the west. The Denali Highway which is 135 miles in length transects the site. The range of vegetation in the test site includes all vegetation associations normally found in interior Alaska. Typical vegetation units are black spruce, alder, willow, and dwarf birch. Tundra units include many varieties of lichens, moss and grass. The test site is in an area which is known as the "Denali," the Athabascan Indian name for Mt. McKinley, meaning "The Big One." The Alaskan Test Site is typical of the northern spruce-tundra biome.

Arizona test site

The Arizona test site is located in the extreme northwest corner of the state of Arizona in the BLM Arizona Strip District. The test site is approximately one million hectares in size. It is bordered on the west and north by the states of Nevada and Utah respectively. Its eastern boundary is range line 6W and 7W and the Colorado River forms its southern terminus. The test site is approximately 88 km southeast of the town of St. George, Utah. The test site characterizes the southwest desert community and includes vegetation types common to that unit; such as, creosote, big sage, and blackbrush. In addition significant stands of pinyon-juniper and ponderosa pine are found in the higher elevations.

Idaho test site

The Idaho test site is in the southwest corner of the state of Idaho and is also comprised of approximately one million hectares. It is bordered on the north by the Snake River, on the east by the Boise Meridian, on the west by the state of Oregon and on the south by the state of Nevada. The test site typifies the extensive sagebrush-grassland communities of the inter mountain basins. Vegetation in the test site is primarily grass and sage with some douglas fir and pinyon-juniper stands in the higher elevations.

RESOURCE INVENTORY REQUIREMENTS

Landsat data processing

BLM is utilizing an interactive digital image analysis system for processing of Landsat data. This system is capable of processing digitized multispectral scanner (MSS) data and has a "stand alone" image processing capability which provides a wide range of digital image processing techniques. The system is configured to utilize a minicomputer and is user interactive, with an interactive color display. The system uses a maximum likelihood algorithm for Landsat classification. The hardware and software configurations are readily expandable for maintaining the system at the state-of-the-art and for meeting unique data requirements.

Preprocessing

BLM applies the following radiometric corrections to the Landsat data. These corrections are applied as appropriate depending upon the task to be accomplished. These radiometric corrections are: line drop, banding removal, noise filtering, atmospheric path radiance, and sun angle.

In addition to the previously mentioned radiometric corrections, the following geometric corrections are applied: mirror scan velocity profile, detector sampling delay, panoramic distortion, scan skew, earth rotation, spacecraft velocity, perspective geometry, attitude, altitude, map projection, and adjustment to ground control. These corrections are applied to obtain an accuracy consistent with the base map.

Classification for vegetation

A maximum likelihood algorithm is preferred for the classification process. BLM uses unsupervised, supervised and controlled clustering techniques for classification. Generally speaking controlled clustering is the approach most commonly used in BLM, due to the spectral variability which is found in wildland environments. Once the digital data has been classified (in this case a final product is referred to) it is desirable to apply a "smoothing" or "cleaning" algorithm to reduce the "salt and pepper" effect of pixel by pixel classification. In addition BLM has a requirement to "generalize" the classified data to provide representation of cover types for each map scale required. The classified data must also be quantified in terms of acreages for combinations of vegetative communities and their characteristics. These acreages must be compiled and tabulated for each class by total area and/or jurisdictional unit.

Data must be environmentally stratified for such parameters as topography, elevation, precipitation, and ecotypes. In addition administrative boundaries must be overlain on the Landsat data to the accuracy of the base maps used in the process. When appropriate BLM performs multidate processing to obtain additional information and accuracy from the classification process. Change detection is utilized to map physical changes resulting from wildfire, mining operations, and the like. It is also necessary to digitally mosaic adjacent Landsat scenes in order to produce a composite image of a study area.

Accuracy determination

In the past the evaluation of the viability of in place vegetation type maps produced through discriminant analysis of digital data image processing technicians have been inhibited by a lack of "a posteriori" verification. Verification provides an index with respect to the utility of the product. This verification must be a statistically valid evaluation of the accuracy of the classification and should be accomplished through a systematic or random selection of pixels from each class on the Landsat product. Pixels can be compared to classifications based on photo interpreted plots and ground samples to determine percent misclassified. Areas to be examined in the verification process must be chosen independently of the areas used in the classification process. The sampling design for verification must account for spatial errors due to pixel misregistration and the realities of collecting photography and ground data for the verification. The following items are acquired by BLM in verifying their classification products:

(1) A percentage accuracy of classification for each cover type class, with a corresponding statistical confidence statement for each percentage.
(2) A numerical summary for each cover type indicating the kinds of classification errors (omission and commission).
(3) A narrative indicating probable causes of misclassification; e.g., spectral similarity, spatial resolution, and phenological phenomena, etc.
(4) A definition of class names for cover types encountered in the classification process through species composition determination.
(5) A statistical confidence statement for each area by cover class determination.

Enhancement for geology

BLM uses both standard and enhanced Landsat products for the purpose of geologic interpretation. Enhanced products are produced using a variety of digital image processing techniques; such as ratioing, contrast stretching, truncation, fourier filtering, etc. Both environmental and geomorphologic data are required. Examples of the parameters of significance from our program in Alaska are as follows:

A. Environmental—structural geology
   1. Active volcanoes
   2. Geothermal hot spring areas
   3. Active glaciers, with aerial extent and whether advancing or retreating
   4. Active landslide areas and potential hazard areas

5. Large active snowslide areas
6. Severe erosion areas—classify genetically
7. Lineaments interpreted to be of geologic, tectonic significance
8. Geologic structures

B. Geomorphologic
1. Morainal features, classified as lateral or terminal
2. Outwash plains, classified as well-drained or poorly drained
3. Eskers
4. Alluvium, classified as silt and clay, gravel, or unsorted
5. Beach deposits
6. Igneous features, which produce a significant landform such as dikes, intrusive igneous structures with a topographic expression, volcanic cones, calderas, lava flows, etc.
7. Karst topography
8. Permafrost and nonpermafrost areas
9. Active strip mining areas
10. Bedrock geology—identify and map those remote sensing units which coincide with conventional geologic mapping units, i.e., formations, members.

Multistage sampling

Statistical sampling schemes, using multistage variable probability sampling for areal estimate of timber volume and rangeland productivity, are being used in this program in Arizona and Idaho. A Landsat digital classification is used as the stratification and small, medium, and large scale photography are used as necessary with supplemental ground work. The BLM program attempts to address rangelands, forest lands, and woodlands.

Rangelands

For each Landsat classified rangeland vegetation type the following parameters are estimated: ground cover, species composition, vegetation condition, forage production, and grazing carrying capacity. Statistics are aggregated by pastures and allotment.

Forest lands and woodlands

For each forest stand identified in the stratification process the following parameters are estimated: size and age class, areal extent, volume, mortality. Statistics are aggregated for each stand.* These same parameters are estimated for woodlands but since the concept of a stand for woodlands is not well defined, aggregation is by allotment.

* A stand is an aggregation of trees or other growth occupying a specific area and sufficiently uniform in species composition, age, arrangement, and condition as to be distinguishable from the forest or other growth on adjacent areas.

Level I	Level II	Level III	Level IV
Forest	Conifer	Dense Conifer	White Spruce
			Black Spruce
			Black/White Spruce
		Open Conifer	White Spruce
			Black Spruce
			Black/White Spruce
	Deciduous	Dense Deciduous	Aspen
			Birch
			Aspen/Birch
			Cottonwood
		Sparse Deciduous	Aspen
			Birch
			Aspen/Birch
			Cottonwood
	Mixed Forest	Deciduous Mix	Birch/Spruce
			Aspen/Spruce
			Cottonwood/Spruce
		Coniferous Mix	Spruce/Birch
			Spruce/Aspen
			Spruce/Cottonwood

Figure 1—Example of provisional classification framework for Alaskan vegetation

Classification Framework and Platforms

For this project BLM has developed classification frameworks for use with remote sensor data. The classification system developed is suitable for use with data from both satellite and aircraft. Figures 1 and 2 are examples of the Provisional Classification framework for Alaska and Arizona respectively. Using Anderson (1976) the multilevel classification system indicates the fact that different sensors provide data at a range of resolutions dependent upon altitude and scale. In general, the following relations are reasonable: Level I—Landsat, Level II—High altitude aircraft, Level III—Medium altitude aircraft, Level IV—Low altitude aircraft. It is the goal of BLM to develop guidelines, based upon the foregoing, which indicate the scales and levels of data suitable for given management requirements. BLM is also attempting to push the different levels of data, such as Landsat, to its maximum extent hoping to obtain good evaluations of the satellite as a mixed level II, III, and IV sensor. Figure 3 illustrates vegetation classification level versus scale and source of data.

Level I	Level II	Level III	Level IV
Woodland/Brushland	Woodland	Juniper - Pinyon	Juniper
			Pinyon
			Juniper - Pinyon
			Pinyon - Juniper
	Brushland	Mountain Chapparal	Manzanita
			Mixed Chapparal
		Oakbrush	Live - Oak - Gambel's Oak
			Mixed Oak
	Riparian	Cottonwood	Cottonwood
			Cottonwood - Salt Cedar
			Cottonwood - Willow
		Tamarix	Salt Cedar
			Salt Cedar - Cottonwood
			Salt Cedar - Willow

Figure 2—Example of provisional classification framework for Arizona vegetatio

Class	Example	Scale	Source
I	Tundra	1:250,000	Landsat
II	Dry Tundra	1:120,000	High Altitude Photography
III	Closed Mat and Cushion	1:30,000	Medium Altitude Photography
IV	Dryas Meadow	1:6,000	Low Altitude Photography

Figure 3—Classification level versus data source

OUTPUT PRODUCT REQUIREMENTS

A requirement exists for a variety of output products. Color coded classified maps adjusted to a Universal Transverse Mercator (UTM) projection are required. These maps are to a scale of 1:250,000 and are aggregated in eight or 32 hectare units as required. Polygon maps showing vegetation as sets of closed polygons are also used. Polygon maps are typically aggregated in eight hectare units. Another product which BLM has found to be beneficial is the line printer map which is typically at a scale of 1:24,000. In this product the aggregations are dependent upon print character size. Other products which are obtained from Landsat data are: geologic maps showing environmental, structural and geomorphological data; surface water maps showing clear and sediment laden water; fire hazard maps showing vegetation types and spread rates.

SUMMARY

BLM is responsible for the multiple use management of the public lands. Public Law requires that a continuing and current inventory be maintained on the public lands and their resource values. In order to meet the requirements of law, as well as to insure that the stewardship of the public lands and their resources leads to the optimum planned use consistent with the long range public good, BLM is evaluating remote sensing as an operational tool for resource inventory.

The remote sensing program and its attendant activities described in this paper attempt to define methodologies to acquire quality resource inventory data. Vegetation units are mapped, acreages are compiled. Multistage sampling techniques are used to estimate timber volume and standing forage. Change detection methodology is used to map physical change from wildfire and mining activities. Geologic maps are produced from sophisticated image processing techniques. All of the resultant data are verified using statistically valid processes. Products are evaluated and interpreted by qualified resource specialists who have also received intensive training in remote sensing. The result is the development within BLM of information requirement guidelines defining platform levels (satellite; high, medium, or low altitude aircraft) suitable for selected resource inventory tasks.

BIBLIOGRAPHY

1. Bonner, Garratt and Torbert Carneggie, Wildland Vegetation Resource Inventory, Bureau of Land Management Detailed Project Plan, 1976.
2. Anderson, Hardy and Witmer Roach, A Land Use and Land Cover Classification System for Use with Remote Sensor Data, Geological Survey Professional Paper 964, 1976.

Digital image analysis techniques required for natural resource inventories*

by WAYNE G. ROHDE
EROS Data Center
Sioux Falls, South Dakota

INTRODUCTION

Landsats-1 and -2 have provided Earth resource scientists the opportunity to acquire multispectral data repetitively over large regions. These data are being used to monitor change in Earth resources, map resources over extensive regions, and inventory specific resource parameters. Application of these data to resource management problems necessitates development of interpretive methodologies that allow quick, consistent, and accurate extraction of pertinent information. In the 1960's, considerable effort was placed on developing digital and analog multispectral processing capabilities. Most of the capabilities were developed through research with aircraft multispectral scanner data. Many initial image processing capabilities were developed on systems that provided little flexibility to the analyst, were difficult to use, and required considerable computer time to complete an analysis. However, these capabilities and the availability of Landsat data has provided the impetus for engineering and development of new, improved, low-cost image analysis systems.[1–8] The trend has been toward more interactive systems that provide users greater flexibility in analyzing multispectral data more rapidly than previously possible.

Evaluation of digital image analysis of Landsat multispectral scanner (MSS) data for mapping and inventorying natural resources have been made.[9–15] These studies and others have led to the development of image analysis techniques that are useful in resource inventory and mapping programs. The purpose of this paper is to discuss analysis techniques that are commonly required for successful application of Landsat MSS data to natural resource inventories.

IMAGE PROCESSING

Radiometric corrections

Landsat MSS data are characterized by several types of radiometric anomalies including bad data lines and points, radiometric striping, and atmospheric scattering effects. Corrections or normalization procedures must be used prior to image classification to minimize the effect of these anomalies on the interpretation and analysis of Landsat data.

Bad data lines, line segments or pixel dropouts are often referred to as intermittent striping problems. Two techniques are commonly used to insert new data in place of bad data. One technique is to replace the bad data line with the preceding line. A second technique is to replace the bad data by interpolating between the brightness values from the pixels in the line before and after the bad data line (Table I). Although the difference between the two techniques seem slight, the latter tends to provide a more accurate representation of the correct brightness value.

Radiometric striping is characterized by variations in film density that should be uniform but often appear as systematic light or dark lines across an image (Figure 1). Radiometric striping introduces anomalous variations in the spectral signatures of resource cover types and will influence classification performance because of, (1) high variances, or (2) lack of adequate training data to characterize signatures of picture elements occurring in lines with striping. This striping is caused by unequal response of the detectors in the MSS to incident electromagnetic energy.

The Landsat MSS consists of six detectors for each of four spectral bands, a total of 24 detectors. Each of these detectors has a slightly different response to the incident electromagnetic radiation (EMR) falling upon it, thus, for the same intensity of EMR, a slightly different output voltage. Specifically, the detectors have different gains and offsets (Figure 2). To provide a uniform response, the six detectors of each band must be calibrated to a known source and corrections applied so that the output of each detector is the same for the same value of incident EMR over the entire useful range of the detectors. Removal of residual differences after calibration constants have been applied is called "destriping."

Several techniques are available to minimize effects of striping. One technique, histogram normalization, calculates the mean brightness value for each detector in each band (Figure 3). A normalization factor is calculated by ratioing the maximum mean to the mean of each detector. The normalization factor for each detector is then multiplied by the

* Some of the original figures in this paper were in color. Color figures can be obtained from the author.

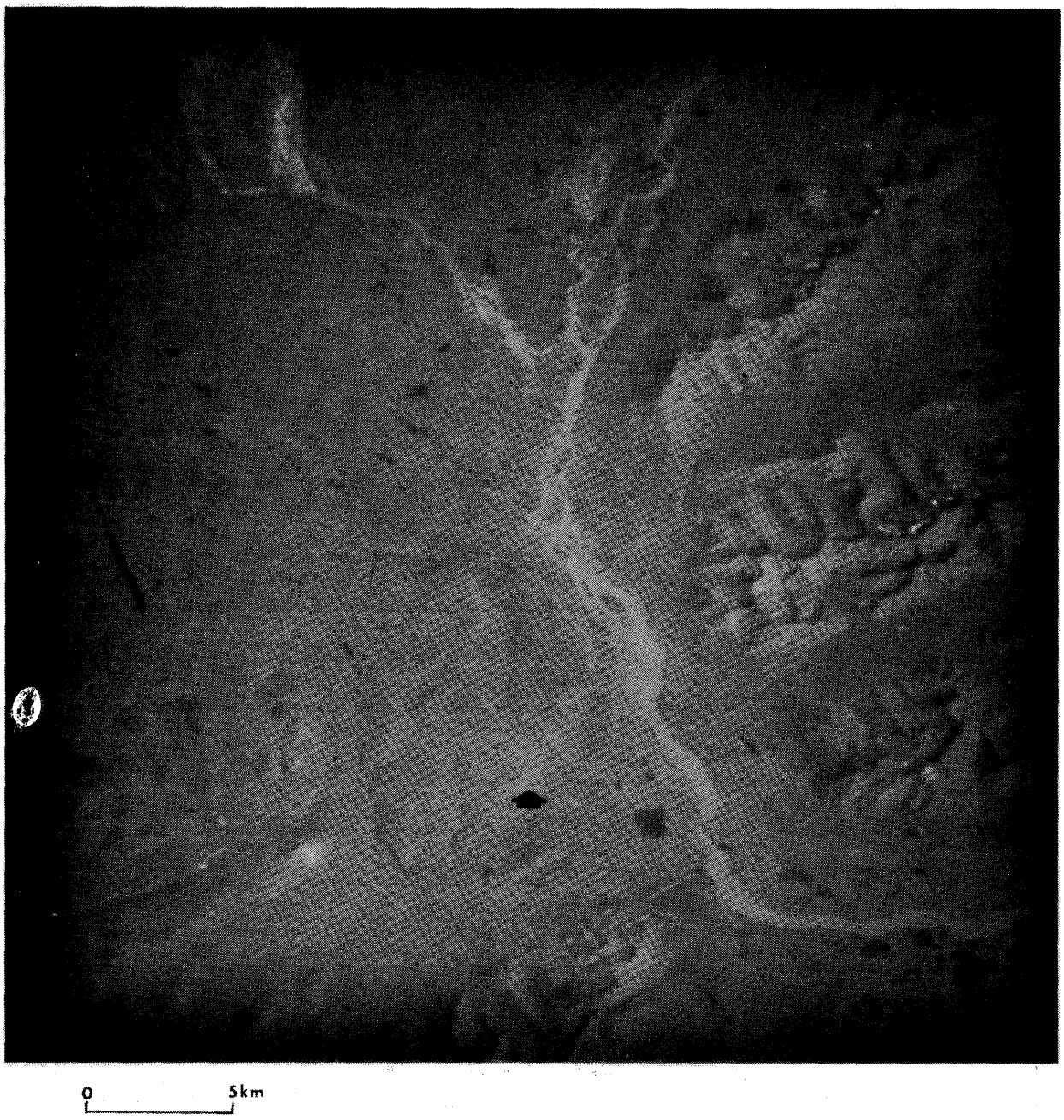


Figure 1—Part of a Landsat color composite from scene 5470-19560 acquired on August 1, 1976 over Cantwell, Alaska. Radiometric striping is characterized by horizontal lines across the image (see arrow). This image has not been geometrically corrected.

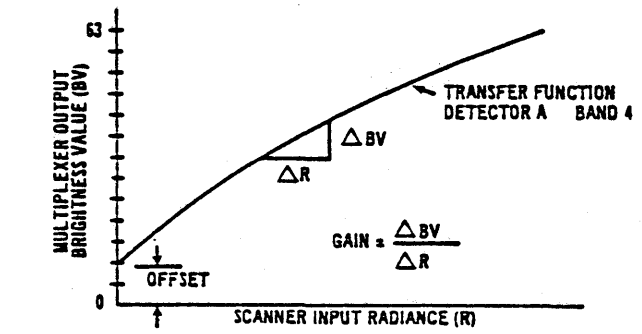
brightness value of each pixel recorded by the detector and the pixel is assigned a new brightness value. Figure 4 shows the same image as Figure 1 after destriping has been performed. Although this technique minimizes striping, residual striping, caused by localized radiometric anomalies, are sometimes found in the data. Other approaches to minimize striping effects in Landsat data are described in Reference 16.

Geometric rectification

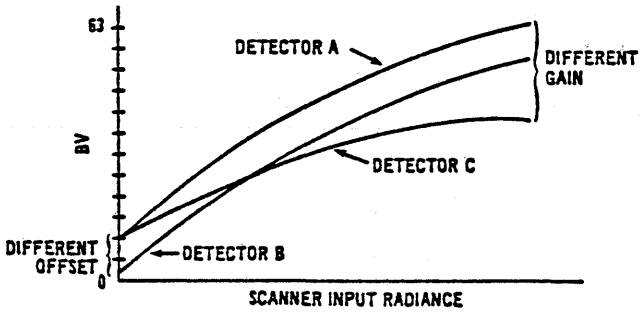
Landsat digital data have inherent geometric errors that must be corrected if the data are to be correctly displayed on film, map overlays, or registered to some geographic reference system. The geometric errors can be either systematic and predictable or variable and measurable.[7] Geometric errors that are systematic and predictable can be

TABLE I.—Brightness values for each of 12 pixels in 15 consecutive scan lines. Line 8, typical of a bad data line, has pixels with brightness values considerably higher than pixels from adjacent scan lines. The bad data line can be corrected by replacing the data with values from corresponding pixels in the preceding line or by interpolating between brightness values from the pixels in the line before and after the bad data line.

	PIXELS											
Lines	1	2	3	4	5	6	7	8	9	10	11	12
1	20	20	19	19	19	20	22	24	24	22	22	20
2	20	29	20	19	20	20	21	25	24	24	27	28
3	20	19	19	20	19	19	22	23	28	28	23	28
4	20	20	19	20	20	21	21	21	26	29	27	26
5	20	19	20	25	25	24	23	23	23	25	29	26
6	19	19	19	20	26	26	26	24	22	22	22	27
7	20	20	20	20	20	20	20	20	22	24	22	26
8	58	58	58	58	58	63	63	58	58	58	62	68
9	20	20	19	20	20	20	22	23	26	26	23	26
10	20	20	23	29	30	29	23	23	27	27	23	21
11	29	29	29	29	30	26	24	23	23	23	25	24
12	22	26	27	27	27	26	24	22	24	24	26	27
13	24	26	24	26	27	28	27	27	24	27	26	26
14	28	25	27	25	25	25	25	27	25	25	25	25
15	23	23	26	29	23	23	23	26	29	26	23	28



DEFINITION OF GAIN AND OFFSET



DIFFERENT GAINS AND OFFSETS FOR 3 BAND 4 DETECTORS IMAGING THE SAME LANDSCAPE CONDITIONS

Figure 2—Causes of radiometric striping in Landsat data

modeled and corrections applied to the data.[7,17] However, errors caused by variations in spacecraft attitude and altitude are not systematic or predictable. In natural resource inventories, accurate location of sample plots or specific resource features are required, thus, the non-systematic errors must also be corrected.

When variable errors are in the data, the effect of the errors can be determined by measuring the apparent displacements of ground control points. Ground control points are features that can be located in an image and whose geographic positions are known.

Once the displacement of control points is known, correction functions or mapping transformations are derived and used to calculate coordinates and map pixels into their correct spatial location on the corrected image. The corrected location of a pixel seldom coincides with a pixel in the distorted image, thus the distorted image must be resampled to determine the brightness value of the pixel in the corrected image. Three commonly used resampling techniques are nearest neighbor, bi-linear interpolation, and cubic convolution. In nearest neighbor resampling, a pixel in the corrected image is assigned the brightness value of the nearest pixel in the uncorrected image.

In bi-linear interpolation resampling, the brightness value of a pixel in the corrected image is determined by interpolating between brightness values of the four nearest pixels in the uncorrected image. In cubic convolution resampling, the brightness value of a pixel in the corrected image is determined by interpolating between the brightness values of 16 nearest pixels in the uncorrected image. More detailed discussions of geometric correction and resampling techniques are discussed in References 8 and 17 through 21.

In most natural resource inventory projects, that have utilized Landsat data, correction functions were calculated prior to classification, but the actual corrections were applied to the classification results. This procedure has been used largely because it is less expensive to correct one band of classification results than to correct four bands of multispectral scanner data.

In addition to correcting Landsat MSS data or classification results to a geographic reference system, geometric correction functions can be used to register one Landsat scene to another, register any graphic or map data to Landsat data, or compute the location of sample units for resource inventory applications. These capabilities are useful in subsequent analysis tasks to perform image stratification, calculate training statistics, refine image classification, allocate sample units, and estimate the geographic coordinates of sample units.

DIGITAL IMAGE CLASSIFICATION

Basically, the purpose of digital image classification is to group a large number of pixels into a number of meaningful categories that can be related to resource features. This grouping is normally done by a classification decision rule (e.g., minimum distance to the mean, parallelepiped, maxi-

mum likelihood). In resource inventory projects, Landsat digital data are commonly analyzed using interactive analysis procedures and a multispectral classification algorithm. Most classification algorithms require the analyst to specify training statistics for the desired resource classes.

Developing training statistics

Fleming, Berkebile, and Hoffer[22] describe three approaches used to derive training set statistics for use with a classification algorithm. These three approaches can be referred to as supervised, unsupervised, and modified or controlled clustering. When developing training statistics, the analyst must ensure that the training statistics represent the variation in spectral characteristics of all cover types found in the project area.

In supervised classification, the analyst selects areas that are spectrally homogeneous for each resource category. Because of variations in spectral characteristics of resource cover types, there often is more than one training area selected for each category. Training statistics, including the mean brightness value within the training area, variance about the mean, and the covariance between data channels are calculated for each training area. Adaptations of this approach use a clustering technique to identify a unique number of spectral classes within each training area. In either case, the analyst assigns a resource cover type to each training area prior to deriving the training statistics.

Unsupervised development of training statistics uses a clustering algorithm to group pixels selected for training into a number of relatively homogeneous spectral classes. The analyst can usually specify certain parameters to the clustering program including the maximum number of classes,

RAW DATA:

DETECTOR	BAND 4	K	BAND 5	K	BAND 6	K	BAND 7	K
1	21.3	1.015	15.5	1.038	29.2	1.084	30.9	1.046
2	20.4	1.064	15.4	1.046	29.8	1.062	28.2	1.146
3	21.2	1.020	16.1	1.000	29.4	1.079	27.9	1.157
4	21.3	1.019	15.3	1.049	29.9	1.060	28.1	1.148
5	21.2	1.023	15.2	1.056	31.7	1.000	27.4	1.179
6	21.7	1.000	15.7	1.028	29.8	1.063	32.3	1.000

NORMALIZED DATA

DETECTOR	BAND 4		BAND 5		BAND 6		BAND 7	
1	21.4		16.2		31.7		32.3	
2	21.5		16.4		31.7		32.3	
3	21.4		16.1		31.7		32.3	
4	21.4		16.3		31.7		32.3	
5	21.5		16.3		31.7		32.3	
6	21.7		16.3		31.7		32.3	

Figure 3—Mean brightness values for each detector in each spectral bank were calculated based on the data from the image shown in Figure 1. Normalization factors (K) are determined by ratioing the mean of each detector and the maximum mean for each band. After data are multiplied by the normalization factor, the normalized means indicate less variability between detector means, thus minimizing the effect of striping.

Figure 4—Part of a Landsat color composite from scene 5470-19560 acquired on August 1, 1976 near Cantwell, Alaska. The data have been destriped using a histogram normalization technique (compare with Figure 1). This image has not been geometrically corrected.

maximum standard deviation within a class, and the minimum number of pixels within a class. In addition to the clustering parameters, the analyst must also specify the pixels to be used in developing the training statistics. He may specify all pixels in the project area or randomly select sample areas of some specified size. Clustering of all pixels in a project area into a small number of spectral clusters or classes is sometimes referred to as unsupervised classification. After unsupervised classification, it is necessary to identify the resource category represented by each cluster. An image showing the distribution of the resource categories is made by combining all clusters of the same resource category into one class.

When classifying wildland vegetation, it is often difficult to identify spectrally homogeneous training areas for each resource category. In such situations, clustering of pixels within randomly selected training areas is often a useful approach for developing training statistics. This approach

was used to develop training statistics for classifying wildland vegetation and other land cover types on a 118,000 ha (293,000 ac) area near Cantwell, Alaska.

The area shown in Figure 4 was gridded into 4,096 eight-pixel-by-eight-pixel cells. Ten percent of the cells were randomly chosen and processed with a clustering algorithm to identify a unique number of spectral clusters in the data. Fifty-six spectral clusters were identified in the sample data. Using this approach to derive training statistics, it was assumed that the sample data represented all of the cover types in the area. Further, it was assumed that each cover type would be represented by at least one spectral cluster and that each spectral cluster would represent only one cover type.

A mean brightness value and its variance for each spectral band and a covariance matrix was calculated for each spectral cluster. These statistics were used in a maximum likelihood algorithm to classify each picture element into one of the 56 cluster classes. Color infrared photographs and field data collected in July 1976, were used to assign each cluster class to one of nine land cover classes. The data were color encoded and recorded on film with a film recording device (Figure 5).

Controlled clustering is another approach to developing training statistics. Using this approach, the analyst selects, either randomly or purposively, polygonal shaped training areas that usually include several resource cover types. Pixels within the training areas are clustered into a number of spectral clusters as in the unsupervised approach. The spectral clusters are printed onto paper or displayed on a color monitor. Using aerial photographs, ground data, and other supporting data, the analyst identifies the resource cover type associated with each spectral cluster prior to performing the maximum likelihood classification.[22]

Controlled clustering was used to develop training statistics to classify the extent of flooded agricultural land in an area near Fargo, North Dakota. Five training areas, each approximately 2,200 ha, were located on color infrared photographs. Areas were selected which represented the range of flood conditions and land cover types throughout the area. The training areas were located on the Landsat data and brightness values of all pixels within the training areas were clustered into a large number of spectral clusters. A line printer map of each training area was compared with a color infrared photographs and each spectral cluster was assigned to one of the ground cover classes shown in Table II. The statistics for all spectral clusters were used to calculate a statistical measure of the separability between spectral clusters in multidimensional space. The separability statistic and the interpreted cover class for each spectral cluster were used to determine which spectral clusters could be combined. The final statistics file contained 21 spectral clusters.

The training area statistics and the Landsat data were put in a maximum likelihood classifier. Evaluation of initial results indicated confusion between older residential areas and agricultural land which had been partly inundated. Fallow fields and fields recently plowed and planted were also misclassified as inundated (Figure 6).

TABLE II.—Land cover classification scheme used in mapping the extent of flooding near Fargo, N. Dak. in July 1975.

Cover type
1. Urban
2. Agricultural land not affected by flood
3. Partially inundated—agricultural land partially affected by flood condition
4. Completely inundated—agricultural land where effects of flooding were apparent over entire field
5. Standing water
6. Wooded (both dry and flooded)

The Landsat digital data were stratified into three strata to minimize the variation in land cover and inundated agricultural land within each stratum. A Landsat color composite was enlarged to a scale of 1:250,000. The general area affected by flooding was delineated on the enlarged Landsat image and plotted onto U.S. Geological Survey 1:250,000-scale maps and digitized. All urban areas greater than approximately 65 ha were plotted on U.S. Geological Survey 1:24,000-scale maps, and digitized. The digitized strata boundaries were registered to the Landsat image with a mean residual error of less than 0.5 pixel.

The strata boundaries were used to extract the image data associated with each strata. The original training statistics were modified to reduce the number of training classes and minimize the likelihood of misclassification noted in the initial results. A separate training statistics file was created for each stratum. The training statistics file and appropriate Landsat data for each stratum were input to a maximum likelihood classification algorithm to classify all picture elements. After classification, the three strata were added to reconstruct the original image. The final classification results are shown in Figure 7. Stratification of Landsat data prior to classification significantly reduced misclassification error.

Summarizing classification results

Image classification results show the areal extent of resource cover types. Because the data are in digital format, the area within each cover class can be easily tabulated (Table III). In most inventory projects, statistical summaries are desired by ownership, administrative units within an ownership, or political boundaries. If these boundaries are plotted onto maps and digitized, the boundaries can be registered to Landsat (Figure 8). The area of each cover type can then be summarized within each defined boundary. This technique has been used in several forest and range resource inventories.[13,23–25] Krebs and Hoffer (1976) have also reported on the use of digitized topographic data to summarize the area of cover types by elevation, slope, and aspect.

SAMPLING FOR ACCURACY ASSESSMENT AND RESOURCE INVENTORY

Although areas of cover classes can be tabulated for any polygonal area within the Landsat data, there is no indica-

Figure 5—Landsat classification of land cover types near Cantwell, Alaska. Barren lands are shown in black. Clear water and sediment-laden rivers are shown in dark blue and light blue, respectively. Tundra is shown in dark green, low shrub is shown in light green, and tall shrub is shown in red. Open conifer-tall shrub is shown in light red, open conifer-low shrub is shown in violet, and dense conifer is shown in purple.

tion of classification accuracy nor can a confidence statement be placed on the tabulated figures. In resource inventories, the parameter of interest may not be measured precisely on Landsat imagery; thus, sampling procedures are needed to derive estimates of the parameter of interest and to evaluate classification accuracy.

Investigators have shown that when remote sensing imagery at several scales is available, it is often most efficient to first make a large number of fast, inexpensive measurements of a parameter (X_i), on small scale, low resolution imagery, and then correlate this parameter with the parameter of interest (Y_i).[12,13,26,27] A second phase involves selecting a small number of sample units on which the parameter of interest (Y_i) is measured precisely on larger scale, higher resolution images. Statistical methods are then used to estimate the parameter of interest from the measurement made on each image type and to estimate the variability about the estimated total.

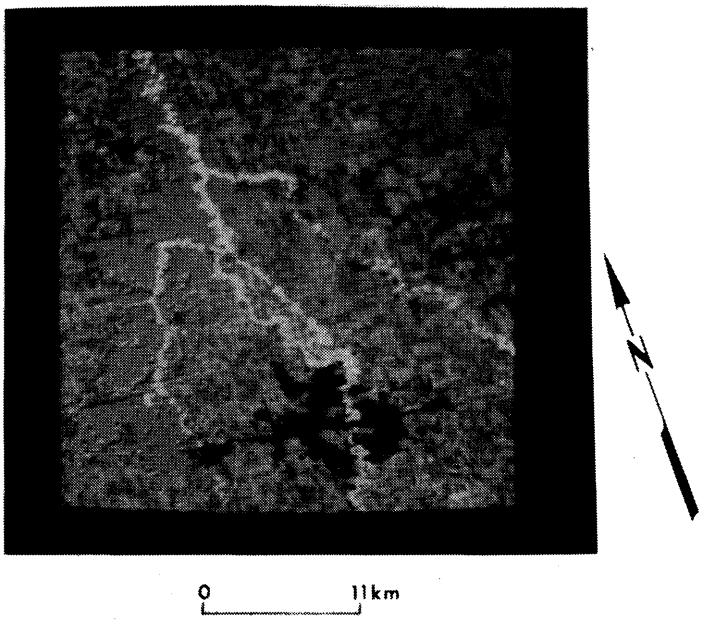


Figure 6.—Landsat classification results showing agricultural land affected by flooding near Fargo, N.D. Wooded areas are shown in light green. Urban areas are shown in red, and cropland not affected by flooding in dark green. Land where complete inundation occurred is shown in dark blue (standing water) and dark brown (land where there was no standing water on July 14, 1975). Land which was partly affected by flooding is shown in yellow. Initial results indicated confusion between older residential areas and agricultural land which had been partly inundated (see arrows).

Verification of classification accuracy

Evaluation of classification accuracy necessitates sampling each of the cover classes. One approach that can be used is a clustered-stratified random sampling technique. This technique will provide the user an estimate of the accuracy of cover types displayed on classification map overlays. This procedure allows evaluating accuracy on a pixel-by-pixel basis and also takes advantage of the efficiency in cluster sampling. To estimate the classification accuracy for the area shown in Figure 5 with an allowable error of ± 5 percent at the .95 probability level, 2,050 individual pixels would have to be sampled. The procedure for estimating sample size is presented by Rohde and others.[28] Because a large number of pixels were to be sampled, cluster sampling was used to reduce the amount of time required to locate plots and interpret specific pixels. Analysis techniques are used to superimpose a sample unit grid on the classification results, randomly select sample units, and tally the results within the selected sample units.

The classified image was gridded into five pixel-by five-line grid cells with each grid cell containing 25 pixels (Figure 9). Each grid cell constitutes a cluster. Sixty-two clusters were randomly selected and plotted on aerial photographs (Figure 10). Each photo was subdivided into 25 subplots (each subplot is equivalent to a pixel).

The predominant cover type at each subplot was interpreted from the aerial photographs. After the photo interpretation was completed, the computer classification of each sample unit was tallied and the classification of each subplot was compared to the photo interpreted cover type. The overall classification accuracy was estimated to be 84.5 percent ± 4.2 percent at the .95 probability level. An accuracy statement was calculated for each cover class using statistics described in Rohde and others.[28]

TABLE III.—Area Classified into Each Cover Class in the Cantwell, Alaska Area Shown in Figure 5.

Cover Type	Hectares	Acres
Clear water	1240	3063
Sediment laden water	2575	6364
Barren land	20508	50672
Tall shrub	7671	18955
Low shrub	26769	66144
Tundra	45524	112488
Open conifer/low shrub	3034	7497
Open conifer/tall shrub	8130	20089
Dense conifer	3159	7805
	118610	293077

Sampling procedures for area estimates

Multispectral classification techniques were used to classify the area of agricultural land near Fargo, North Dakota that was affected by flooding (Figure 7). Although the area could be easily tallied, a sampling procedure was required to place a confidence interval around the estimate.

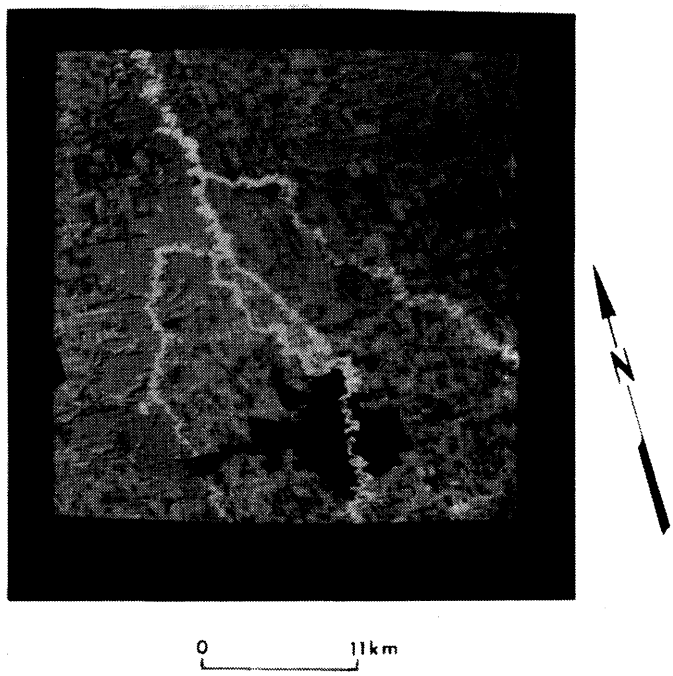


Figure 7.—Landsat classification results showing agricultural land affected by flooding near Fargo, N.D. Image stratification prior to classification reduced the misclassification between residential areas and agricultural land which had been partly inundated (compare with figure 6).

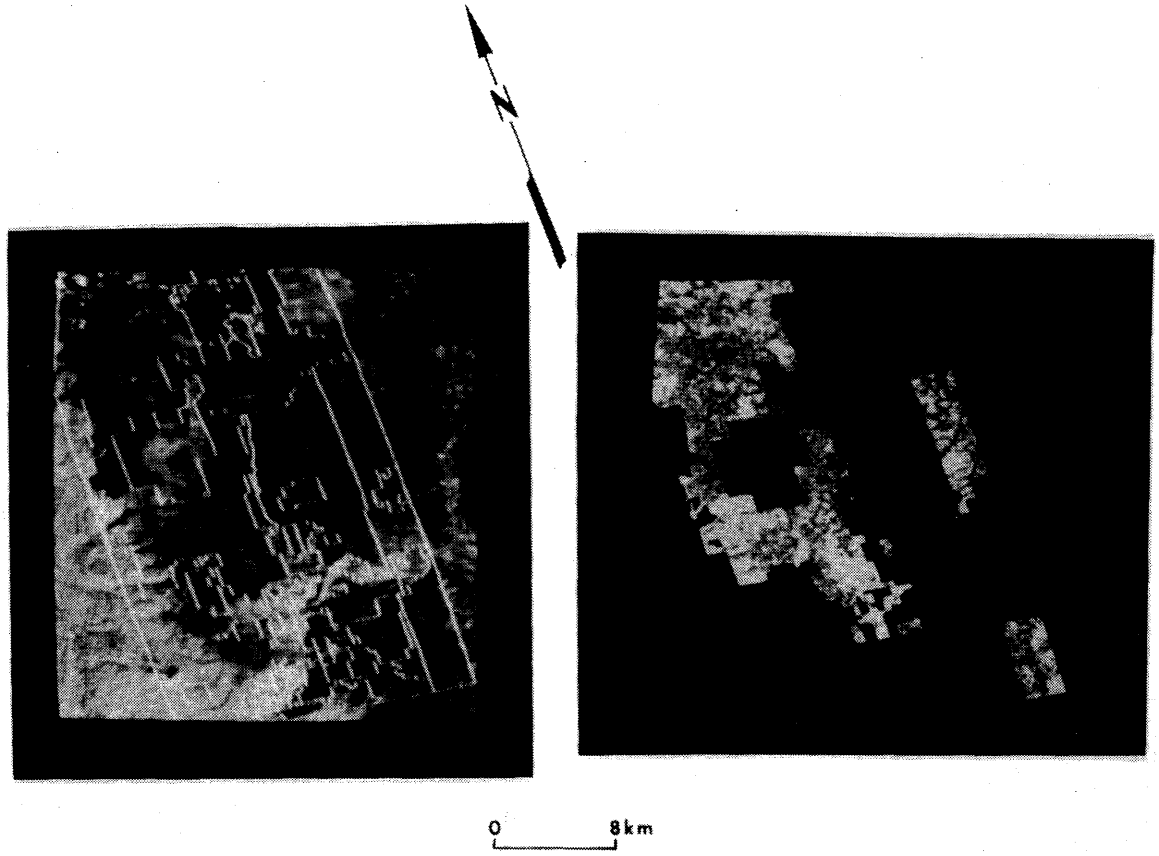


Figure 8.—Part of a Landsat color composite image from scene 2195-17520 acquired on August 5, 1975 (left). Land ownership boundaries were digitized and registered to the Landsat image. The Landsat data were classified into three broad volume classes (low, moderate, high) and summarized by ownership. Classification results for U.S. Forest Service land is shown on the right. Low volume is shown in white, moderate volume in red, and high volume is shown in green.

A two-phase sampling scheme was used. In two-phase sampling, sample units are the same size in each phase. The Landsat classification results provided the quick estimates in the first phase. A subsample of the first-phase sample units was selected for more precise measurements in the second phase. A least squares regression of estimates from the second phase on estimates from the first phase was used to adjust estimates made from the first phase.

Assuming a sample unit size of 1,600 pixels, the number of sample units required for measurement at each phase was estimated, based on an expected correlation between measurements made at each phase, the cost ratio of obtaining measurements at each phase, and the desired accuracy level. To achieve an estimate with an accuracy of ± 10 percent at the .95 probability level, it was estimated that 200 first-phase samples and 30 second-phase samples would have to be measured.[29] A forty-pixel-by-forty-line grid was placed over the classification results and two hundred sample units, 1,600 pixels in size, were randomly selected. The Landsat classification results in each selected sample unit were tabulated to provide the first-phase estimates. Thirty second-phase sample units were randomly selected from the 200 first-phase sample units. The latitude and longitude coordinates were computed for the four corner points of each second-phase sample unit. Each second-phase sample unit was plotted onto U.S. Geological Survey 1:24,000 scaled maps and transferred to color infrared photographs. The area of cropland affected by flooding in each second-phase sample unit was estimated with a dot grid. The area estimates made on the second-phase sample units were used to develop regression coefficients to adjust estimates made in the first phase from the Landsat classification.

Using two-phase sampling statistics, 379,165 ha of agricultural land were estimated to be affected by flood conditions. It was also estimated the 148,020 ha were partially affected and 236,835 ha were completely flooded. The area estimates and associated sampling errors are shown in Table IV.

The ability to overlay a sample unit grid of varying size on Landsat data, tabulate classification results within sample units, and calculate geographic coordinates of sample units, provide data that resource specialists can use to define sampling schemes or allocate sampling efforts to derive resource estimates. Landsat data, multi-phase and multi-stage sampling techniques have been used in numerous resource inventory projects.[28]

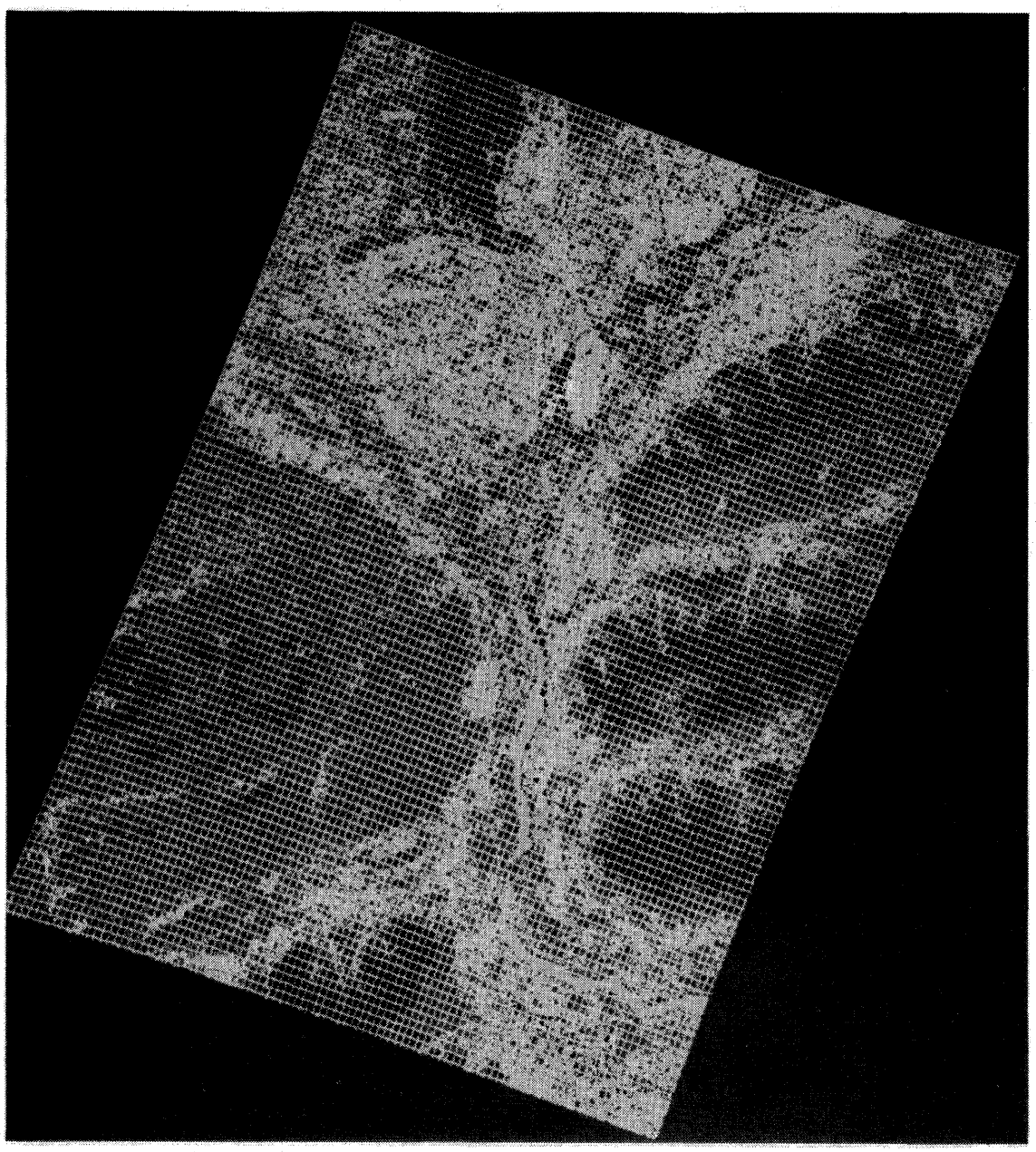


Figure 9.—Five pixel by five line sample unit grid superimposed on Landsat classification of land cover types near Cantwell, Alaska. The color scheme is listed in figure 2. Sixty-two sample units were randomly selected with equal probability for interpretation on aerial photographs to estimate the classification accuracy.

OUTPUT PRODUCTS

Resource managers often need map overlays that show the areal extent of resources being inventoried or mapped. Common requirements include color encoded map overlays or line map overlays at various scales from 1:24,000 to 1:250,000. Digital classification results provide great flexibility in displaying data in various formats and map scales. The data can be easily aggregated for display at various levels of generalization or at a given level of detail to simulate various minimum mapping units.

In most multispectral classification techniques, each individual picture element (0.40 ha or 1.1 ac) is assigned to one class. This often results in a "salt-and-pepper" appearance on map overlays. Spatial smoothing programs can be used to simulate a minimum mapping unit and to reduce the "salt-and-pepper" appearance. The image in Figure 11 was processed with a smoothing algorithm to simulate an approximate 4 ha (10 ac) minimum mapping unit. A three pixel by three line cell was passed through the data one pixel at a time. The first and last lines and the first and last pixels in each line were not changed. All other pixels were reas-

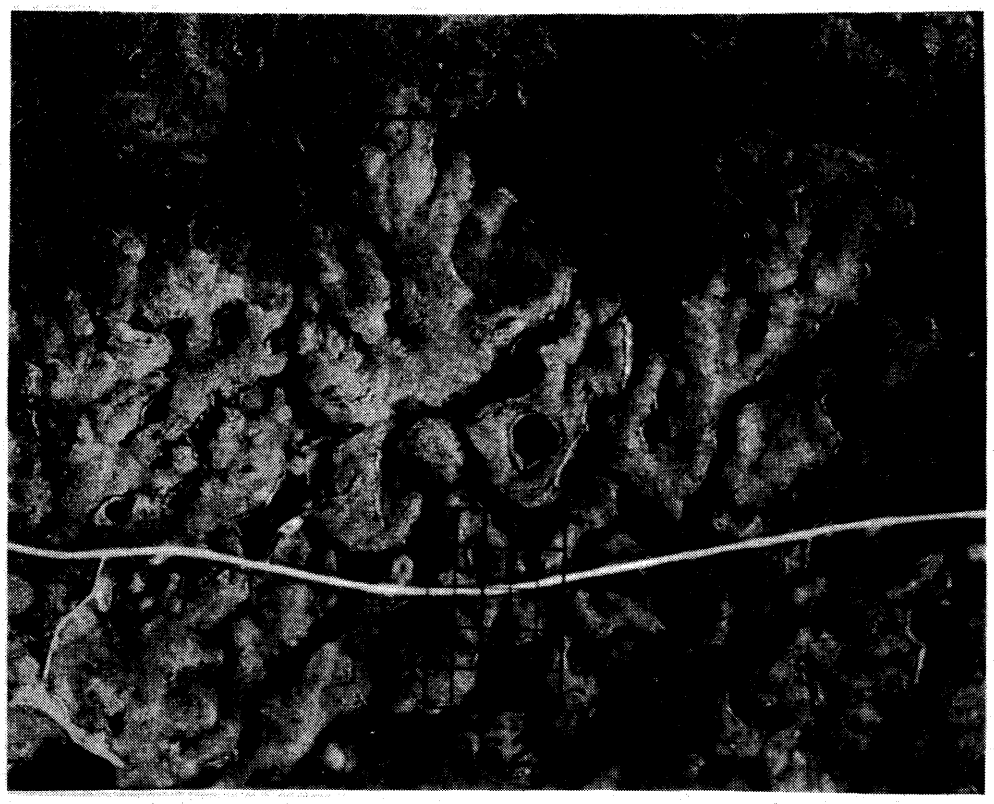


Figure 10.—Color infrared aerial photograph showing sample unit number 4962. Each sample unit was divided into 25 subplots where each subplot represents a pixel (.40 ha or 1.1 ac).

signed to the most frequently occurring class in the three pixel by three line cell. Although this technique tends to reduce misclassification caused by edge effects from several resource types, small, narrow features may be lost unless classes representing such features are more heavily weighted when the smoothing function is applied.

The final classification results can be color encoded on film, printed by a line printer, or plotted onto a map overlay with a plotter. Color encoded map overlays are made by recording onto film the classified digital data that have been geometrically corrected, registered and scaled to the desired map base. The data can be recorded either directly onto color film or onto three separate black-and-white images that can be composited into a color image (Figure 11).

Classification results can be printed with a line printer where each cover class is represented by a unique symbol.

TABLE IV.—Estimate of Agricultural Land Affected by Flood Conditions.

Cover Class	Hectares	Sampling Error
		Percent
Partial inundation	148,020	5.4
Complete inundation	236,835	6.1
All areas affected by flooding	379,165	3.9

With appropriate geometric correction functions, the data can be printed at various scales; the most common are 1:24,000, 1:62,500, and 1:63,360.

Classification results can be plotted as polygons on map overlays that have been registered and scaled to the desired map base. The final map overlay can show each polygon in color with different shading patterns or with polygons numbered to correspond to a cover class in the legend. Figure 12 is an example of a map overlay that corresponds to a portion of a U.S. Geological Survey 1:63,360 scaled map.

Other techniques exist to format classification results to a user specified grid cell size and register the data to an existing information system. This capability allows a resource scientist to merge other ancillary data (e.g., soils, landforms, hydrology, elevation, slope, aspect) with the classification results.

SUMMARY

Landsat digital data with supporting aerial photographs and sampling techniques were used to accurately map wildland vegetation and inventory flooded agricultural land. Image analysis techniques were required to make radiometric and geometric corrections. Techniques were needed to stratify the Landsat data prior to classification to reduce the probability of misclassification.

Interactive analysis techniques were used to develop training statistics and to evaluate initial classification results. The classification results were used in sampling schemes to evaluate classification accuracy and to inventory the area within each cover type. Analysis techniques were used to partition the project areas into sample units, summarize classification results within sample units, and to select and locate sample units. Image analysis techniques were also used to plot the classification results on map overlays that were registered to 1:63,360 scaled maps.

Continued development and improvement of image analysis techniques are needed to make them more efficient. As these techniques are improved, Landsat data, used in its proper context with supporting data, and sound sampling strategies, can be an effective tool in conducting natural resource inventories.

Figure 11.—Landsat classification of land cover types near Cantwell, Alaska. A spatial smoothing algorithm was used to eliminate the "Salt-and-pepper" appearance as shown in figure 5. The smoothing was done to simulate an approximate four ha (ten ac) minimum mapping unit. Barren lands are shown in black. Clear water and sediment laden rivers are shown in dark green; low shrub is shown in light green, and tall shrub is shown in red. Open conifer—tall shrub is shown in light red; open conifer—low shrub is shown in violet, and dense conifer is shown in purple.

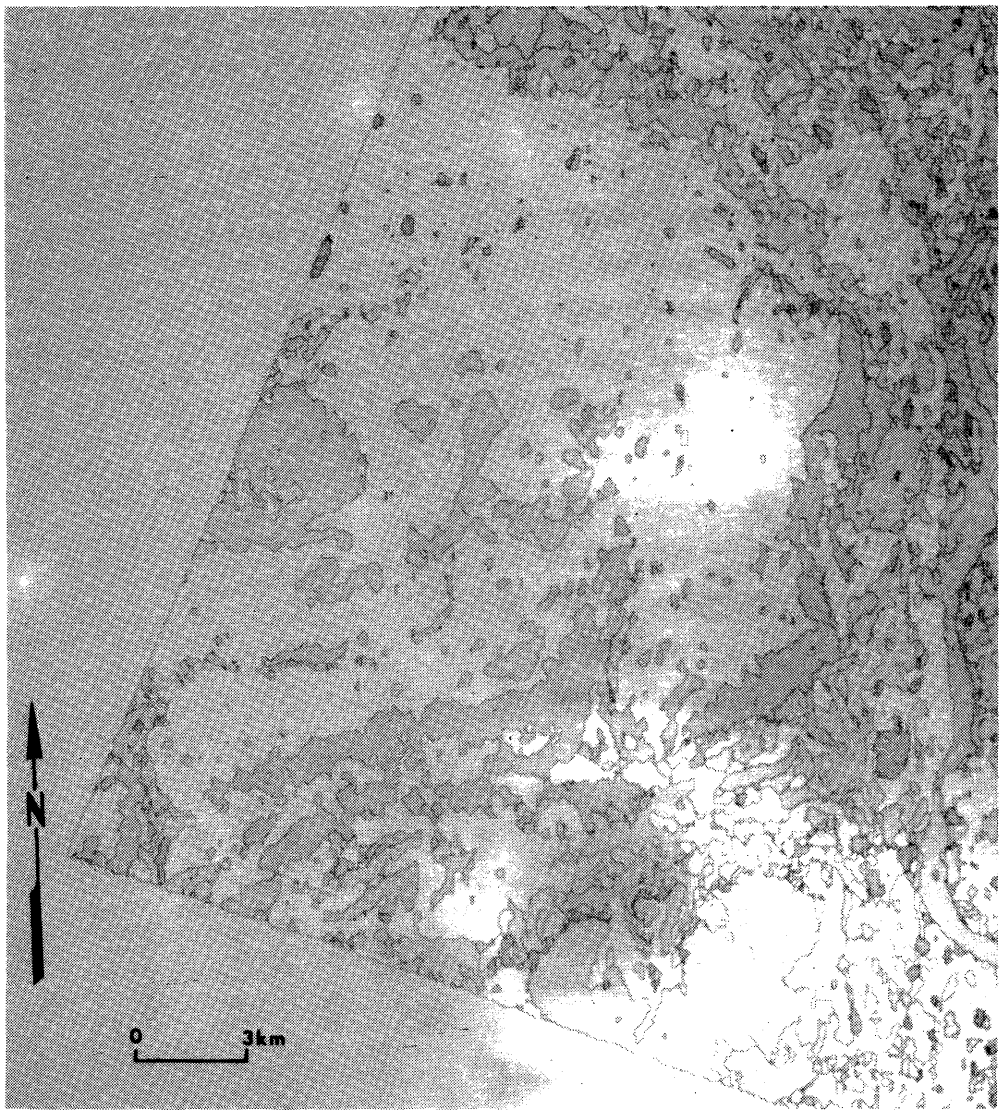


Figure 12.—Example of Landsat classification results that have been geometrically corrected and plotted on a map overlay. This overlay is part of the total area shown in figure 5. The map overlay was scaled to register to the U.S. Geological Survey 1:63,360 Healy A2 map sheet. Barren land is shown in blue, single-horizontal lines. Clear water and sediment-laden rivers are shown in dark blue, double-horizontal lines and blue dots, respectively. Tundra is shown in white, low shrub is shown as green dots, and tall shrub is shown in red dots. Open conifer—tall shrub is shown as green triangles, open conifer—low shrub is shown as dark red, double-vertical lines, and dense conifer is shown as dark red single-vertical lines.

REFERENCES

1. Erikson, J. D., "Advances in Automatic Extraction of Earth Resources Information from Multispectral Scanner Data, in *NASA Earth Resources Survey Symposium,* Houston, Texas, June 1975, Proc.: NASA, Johnson Space Center, Houston, Texas, Vol. 1-B, pp. 1245-1276.
2. Schaller, E. S., and R. W. Towles, "Image 100—The Interactive Multispectral Image Processing System," in *NASA Earth Resources Survey Symposium,* TEXAS, June 1975, Proc.: Houston, Texas, NASA, Johnson Space Center, Vol. 1-B, pp. 1275-1292.
3. Johnson, R. H., "M-DAS—System for Multispectral Data Analysis," in *NASA Earth Resources Survey Symposium,* Houston, Texas, June 1975, Proc.: NASA, Johnson Space Center, Houston, Texas, Vol. 1-B, p. 1323-1350.
4. Quinn, M. J., 1975, "ERIPS—Earth Resource Interactive Processing System," in *NASA Earth Resources Survey Symposium,* Houston, Texas, June 1975, Proc.: NASA Johnson Space Center, Houston, Texas, Vol. 1-B, pp. 1351-1354.
5. Whitley, S. L., "Low Cost Data Analysis Systems for Processing Multispectral Scanner Data," in *NASA Earth Resources Survey Symposium,* Texas, June 1975, Proc.: Houston, Tex., NASA Johnson Space Center, Vol. 1-B, pp. 1355-1376.
6. Henze, J., and R. DeZur, "Interactive Digital Image Manipulation System," in *NASA Earth Resources Survey Symposium,* Houston, Texas, June 1975, Proc.: NASA, Johnson Space Center, Houston, Texas, Vol. 1-B, 1975, pp. 1415-1435.
7. Bernstein, Ralph, and D. G. Ferneyhough, Jr., "Digital Image Processing," *Photogrammetric Engineering,* Vol. 41, 1975, pp. 1465-1476.

8. Goetz, A. F. H., F. C. Billingsley, A. R. Gillespie, M. S. Abrams, R. L. Squires, E. M. Shoemaker, I. Lucchitta, and D. P. Elston, Applications of ERTS Images and Image Processing to Regional Geologic Problems and Geologic Mapping in Northern Arizona, JPL Technical Report 32-1597, Jet Propulsion Laboratory, Pasadena, California, 1975, 188 p.
9. Heller, R. C., R. C. Aldrich, R. W. Dana, R. S. Driscoll, R. E. Francis, W. J. Greentree, R. J. Myhre, N. X. Norick, E. H. Roberts, and T. H. Waite, Evaluation of ERTS-1 Data for Forest and Rangeland Surveys, U.S. Forest Service, Pacific Southwest Forest and Range Experiment Station, Research Paper PSW-112, 1975, 67 p.
10. Hoffer, R. M., Natural Resource Mapping in Mountainous Terrain by Computer Analysis of ERTS-1 Satellite Data, Purdue University, Laboratory for Applications of Remote Sensing, LARS Research Bull. 919, 1975, 124 p.
11. Bentley, R. G., Jr., B. C. Salmon-Drexler, W. J. Bonner, and R. K. Vincent, A Landsat Study of Ephemeral and Perennial Rangeland Vegetation and Soil, Bureau of Land Management, Denver, Colorado, Final Report Type III, March 1975-December 1976, 234 p.
12. Nichols, J. D., M. J. Gialdini, B. Orme, and D. T. Lauer, ERTS as an Aid in Timber Volume Inventory: A Special Report to Office of Applications," NASA Headquarters, Remote Sensing Research Program, Univ. of California, Berkeley, Calif., 1974, 31 p.
13. Gialdini, M. J., S. Titus, J. D. Nichols, and Thomas, R. W., "The Integration of Manual and Automatic Image Analysis Techniques with Supporting Ground Data in a Multistage Sampling Framework for Timber Resource Inventories: Three Examples," in *NASA Earth Resources Survey Symposium,* Houston, Texas, June 1975, Proc.: NASA, Johnson Space Center, Houston, Texas, Vol. I-B, pp. 1377-1388.
14. Krebs, P. V., and R. H. Hoffer, Multiple Resource Evaluation of Region 2 U.S. Forest Service Lands Utilizing Landsat MSS Data, NASA Goddard Space Flight Center, Greenbelt, MD., Final Report Type III, July 1976, 298 p.
15. Rohde, W. G., "Potential Applications of Satellite Imagery in Natural Resource Inventories," in *Integrated Inventories of Renewable Natural Resources,* Arizona, January 1978, Proc.: Tucson, Arizona, (in press).
16. Rohde, W. G., J. K. Lo, and R. A. Pohl, "EROS Data Center Landsat Digital Enhancement Techniques and Imagery Availability," *Canadian Journal of Remote Sensing,* 1977 (in press).
17. Anuta, P. E., Geometric Correction of ERTS-1 Digital Multispectral Scanner Data, Laboratory for Applications of Remote Sensing, Purdue University, LARS Information Note 103073, 1973, 22 p.
18. Bernstein, Ralph, "Scene Correction (Precision Processing) of ERTS Sensor Data Using Digital Image Processing Techniques," in *Earth Resources Technology Satellite-1 Symposium,* 3rd, Washington, D.C., December 1973, Proc.: NASA Goddard Space Flight Center, Greenbelt, Maryland, NASA SP-351, Vol. 1, pp. 1909-1928.
19. ——, "Results of Precision Processing (Scene Correction) of ERTS-1 Images Using Digital Image Processing Techniques," in *Symposium on Significant Results Obtained From the Earth Resources Technology Satellite-1,* New Carrollton, Maryland, March 1973, Proc.: NASA Goddard Space Flight Center, Greenbelt, Maryland, NASA SP-327, Vol. 1, 1973, pp. 1143-1150.
20. Peet, F. G., A. R. Mack, and L. S. Crosson, "Affine Transformations from Aerial Photos to Computer Compatible Tapes," in *Earth Resources Technology Satellite-1 Symposium,* 3rd, Washington, D.C., December 1973, Proc.: NASA Goddard Space Flight Center, Greenbelt, Md., NASA SP-351, Vol. 1, pp. 1719-1724.
21. Rifman, S. S., "Digital Rectification of ERTS Multispectral Imagery," in *Symposium on Significant Results Obtained from the Earth Resources Technology Satellite-1,* New Carrollton, Maryland, March 1973, Proc.: NASA Goddard Space Flight Center, Greenbelt, Md., NASA SP-327, Vol. 1, pp. 1131-1142.
22. Fleming, M. D., S. S. Berkebile, and R. M. Hoffer, Computer-Aided Analysis of Landsat-1 MSS Data: A Comparison of Three Approaches, Including a "Modified Clustering" Approach, Purdue University, Laboratory for Applications of Remote Sensing, LARS Information Note 072475, 1975, 9 p.
23. Nichols, J. D., M. Gialdini, and S. Jaakkola, "A Timber Inventory Based upon Manual and Automated Analysis of ERTS-1 and Supporting Aircraft Data using Multistage Probability Sampling," in *Earth Resources Technology Satellite-1 Symposium,* Washington, D.C., 3rd, December 1973, Proc.: NASA Goddard Space Flight Center, DP-351, Vol. I, pp. 145-157.
24. Nichols, J. D., R. A. Harding, R. B. Scott, and J. R. Edwards, "Forest Inventory of Western Washington by Satellite Multistage Sampling," in *American Society of Photogrammetry,* Fall Convention, Seattle, Washington, September 1976, Proc.: American Society of Photogrammetry, pp. 180-217.
25. DeGloria, S. D., S. J. Daus, R. W. Thomas, and D. M. Carneggie, Spacecraft and Aircraft Remote Sensing for Integrated Unit Resource Inventory and Analysis in Northeastern California, Final Report, 1975.
26. Langley, P. G., "New Multistage Sampling Techniques using Space and Aircraft Imagery for Forest Inventory," in *Internat. Symposium on Remote Sensing of Environment,* Michigan, 6th 1969, Proc.: Ann Arbor, Mich., Michigan Univ., Vol. II, 1969, pp. 1179-1192.
27. Heller, R. C., and J. F. Wear, "Sampling Forest Insect Epidemics with Color Films," in *Internat. Symposium on Remote Sensing of Environment,* Michigan, 6th, 1969, Proc.: Ann Arbor, Mich., Michigan Univ., Vol. II, pp. 1157-1178.
28. Rohde, W. G., W. J. Bonner, W. A. Miller, and C. A. Nelson, Classification of Wildland Vegetation near Cantwell, Alaska with Landsat Digital Data, presented at Annual American Society of Photogrammetry Engineering and Remote Sensing, Washington, D.C., March 1978.
29. Rohde, W. G., J. V. Taranik, and C. A. Nelson, "Inventory and Mapping of Flood Inundation using Interactive Digital Image Analysis Techniques," in *Annual William T. Pecora Memorial Symposium,* South Dakota, 2nd, October 1976, Proc.: Sioux Falls, South Dakota, pp. 131-143.

Digital image analysis applications in state natural resource agencies

by PAUL A. TESSAR
National Conference of State Legislatures
Denver, Colorado

INTRODUCTION

State government agencies are becoming increasingly involved in developing and applying digital image processing capabilities to fulfill state responsibilities. Experimental visual Landsat applications were attempted by many state agencies from the launch of Landsat-1 in July, 1972. Digital applications, however, were generally not attempted in state agencies until the middle of 1974. Since then, many states have been involved in investigations and demonstration projects utilizing digital image processing techniques. Several states (e.g., Texas, Georgia, and South Dakota) have established in-house operational or quasi-operational digital Landsat capabilities. It is clear that the trend of the future is toward the establishment of such capabilities in many states.

Often, the development of digital Landsat analysis and application capabilities is within the context of a state natural resource information system. Digital Landsat data is extremely compatible with geographic information systems, once compensation is made for technical problems. Existence of a state system provides a focal point for adopting satellite remote sensing techniques, thereby facilitating technology transfer efforts. On the other hand, in states which do not have a natural resource information system, the development of Landsat digital analysis capabilities has often led to the examination of more general computerized natural resource data analysis systems. In either case, there is a synergistic effect from the mixing and comparing of several types of data in a computer.

The experiences of two state programs in developing or acquiring and applying digital Landsat analysis capabilities are discussed below. The approach, emphasis and results are quite different in each case. All fifty states, however, are different in terms of institutional structure, political climate, role of state government and local needs. While each state is unique overall, a number of features and experiences are shared by many. By studying similar type developments in other states before attempting them at home, pitfalls can be avoided and successes repeated. Hopefully, the experiences of North Dakota and South Dakota will prove useful in this vein.

THE NORTH DAKOTA REGIONAL ENVIRONMENTAL ASSESSMENT PROGRAM'S STATE LAND COVER ANALYSIS

The North Dakota Regional Environmental Assessment Program (REAP), a division of the State Legislative Council, was created by the 1975 North Dakota Legislative Assembly as a result of a recognized need for a comprehensive and coordinated source of natural resource-related information. The immediate impetus for such a program was current and prospective large-scale coal development in southwestern North Dakota. Enabling legislation directed REAP to carry out studies and research related to natural resources and to develop a comprehensive information and analysis system for the storage, retrieval, and analysis of such information. Furthermore, REAP was especially directed to develop the capability of forecasting the impact of natural resource development alternatives which may be considered by the state.

In order to meet its responsibilities, REAP has undertaken four specific tasks: (1) the collection of existing baseline data, together with a major effort to fill recognized data gaps; (2) the design, development, and installation of a computer-based information system capable of storage, retrieval, representation, and analysis of appropriate data; (3) development of a variety of capabilities, including models, for assessing the impacts of proposed natural resource development; (4) the establishment of a means for the regular monitoring and updating of data so the information system will remain current and the analyses relevant.

In the fall of 1975, REAP conducted an exhaustive study of the relevant data sources and gaps. Through this effort, it was recognized that there was no comprehensive land cover inventory for the state of North Dakota. Accordingly, REAP identified the completion of such an inventory as a major task requiring early attention.[1]

Several different technologies were consideered for inventorying the land over the entire state of North Dakota, approximately 180,000 sq. km. (70,665 sq. mi.) in area. The technology chosen, satellite remote sensing, uses imagery obtained from the Landsat satellites launched by the National Aeronautics and Space Administration in July 1972

and January 1975. One of the major advantages of Landsat imagery is that the entire state of North Dakota can potentially be covered every nine days, and therefore repetitive comparative imagery is available. This technology therefore provides a mechanism for conducting a baseline land cover analysis for the state and for monitoring how that land cover changes over time. Since a near-permanent platform for imaging is available at no cost to the state, it is more economical to use such a system than to undertake complete aerial photographic coverage of the state, and to propose frequent repetition of the coverage for monitoring purposes. Another distinct advantage is the large areal coverage for any given image which permits discerning gross land features often not perceivable with aerial photography. Landsat imagery is particularly amenable to computer analysis, whereas such techniques have not been well developed for photographic imagery. A disadvantage of utilizing Landsat imagery is that a high degree of detail, particularly in urban-type areas, is usually not available. Thus, it is generally not feasible to attempt to obtain a full level II land cover analysis utilizing Landsat imagery; a thorough level I analysis is readily obtainable, however.[2]

Categorization

Computer processing of the Landsat data from computer tapes as received from the EROS Data Center was performed by the Bendix Aerospace Systems Division using a Bendix datagrid digitizer system and the Bendix multi-spectral data analysis system (MDAS).[3] The process involved identifying an area on a computer display terminal which coincided with the exact shape and location of a given training set. A computer was then coded to identify all areas in that particular scene with a similar reflectance. Finally, a color was assigned to indicate all such areas. In a follow-up process, color-coded areas of a particular category were checked against the available ground truthing information to ensure that the computer search for similar cover was complete. This process was repeated for each of the land cover categories. The final result was what is called a "categorized" tape.

As a result of categorization of the 19 Landsat scenes, as few as 12 and as many as 36 different categorizations were determined. For example, in two different scenes up to 14 different categories of water were identified. As many as nine different categories of cropland were detected but there were only three different wetland categories. All categories identified through this process have been retained on the categorized tapes. However, for purposes of displaying the results of this land cover analysis on colored maps, an extensive merging of categories was undertaken.

Ten different land cover categories were selected for display on maps and a color was assigned for each category. The 10 categories selected for map display include all seven applicable Level I categories suggested by the U.S. Geological Survey,[2] and five Level II and Level III sub-categories.

Results

North Dakota is one of the first states for which detailed land cover analysis has been completed utilizing computer processing of digital Landsat imagery. The results of this land cover analysis have been incorporated in several different products which will be available for public purchase and technical use. Each of these products will be described in some detail below.

The first product of the land cover analysis is a color-coded map for each of the 53 North Dakota counties at a scale of 1:126,720 (exactly two miles to the inch). On each map, a color is used to designate each of the 10 different land cover categories as indicated in Table I. Each county map contains information on the scale, the color legend, the name of the county, a small map of the state with the county darkened, and an outline of the county containing the date(s) of the Landsat imagery used for the analysis. To produce these maps Bendix developed a technique to create a high quality color negative merging the land cover and the base map data. The land cover data was produced by the computer processing discussed above, and the base map was an edited version of the detailed county highway maps.

The second product of this analysis is an area tabulation for each township, county, and the state. For each of these geographic units, a table was prepared indicating the acreage covered by each of the 10 land cover analysis categories. The area tabulations were prepared by reading a digital tape file and summing the land cover to the appropriate geographic area. The area tabulation for the entire state is shown in Table I.

The third product of the land cover analysis is a map of the entire state of North Dakota at a 1:500,000 scale (approximately eight miles to the inch) which contains the land cover illustrated by the appropriate color, the same as used on the county maps, a legend and the county boundaries.

The final product of the land cover analysis is a digital tape file of land cover data prepared in a specified cell format. The objective of this effort was to prepare land cover information aggregated to 40-acre cells, registered by

TABLE I—North Dakota Land Cover by Category

Land Cover Category	Acres	Percent of Total
Built-up Land	19,777	0.04
Cropland	15,956,690	35.24
Fallow Land	6,189,168	13.67
Exposed Subsoil or Saline Seep	100,952	0.22
Rangeland	11,889,387	26.26
Mixed Range, Agriculture, and Pasture	7,630,329	16.85
Forest	386,823	0.85
Water	1,265,812	2.80
Wetland	780,284	1.72
Barren Land	975,863	2.16
Uncategorized	80,787	0.18
Total	45,275,872	99.99

quarter-mile section of the Public Land Survey. Thus, all pixels with center points lying within the given quarter-quarter section were tabulated and grouped by land cover category.

Summary

The first complete land cover analysis of the state of North Dakota has been conducted utilizing computerized analysis of Landsat imagery. The products of the analysis are color-coded maps of each of the 53 counties in the state and a state map. Tables of area coverage of each land cover category are also available. The digital information for each pixel in the state is recorded on computer tape for subsequent utilization in the REAP computer system.

This analysis has revealed a number of unique capabilities of the Landsat imagery process, those utilizing more than a single Landsat scene. The products of this analysis are already being used in corridor analysis for transmission lines, resource analysis, water management problems, non-point source pollution studies, and regional environmental analysis. The results provide an essential data base for the REAP system and will serve as the basis for monitoring changes in land cover in the state of North Dakota.

LANDSAT APPLICATIONS OF THE SOUTH DAKOTA LAND RESOURCE INFORMATION SYSTEM

In the early 1970's, the South Dakota State Legislature recognized the need for land related policy and planning decisions to be based on solid, objective data. In 1974, the Legislature facilitated the establishment of this information base by authorizing the South Dakota State Planning Bureau to investigate the means for collecting the desired land resource data. This investigation resulted in the initiation of the South Dakota Land Use Inventory and the formulation of the Land Resource Information System (LRIS). The inventory, using both photographic and digital Landsat data, began as a three-year, three-phase cooperative project between the State Planning Bureau and the EROS Data Center. The agreement specified that EROS would supply technical assistance and imagery, while the state would supply personnel to initiate and maintain an in-house system for interpreting, storing and applying the land use data to South Dakota operations.

One important factor regarding the South Dakota Land Use Inventory needs to be emphasized. The State Land Use Inventory program established an in-house analysis capability that will allow mapping as the need arises. The ability to perform special manipulations with existing data, the tools for updating the inventory, and the capability for special projects such as the 208 work make the program especially flexible and valuable to the state. Many of the applications outlined below have resulted from ad hoc projects that necessitated some form of additional manipulation of analysis of the standard county land use data.

Phase I—Land use analysis

The initial phase of the South Dakota Land Use Inventory produced a map displaying a generalized picture of the distribution of major land uses across the state. Seventeen Landsat scenes from the spring of 1973 were required to provide cloud-free coverage of the entire state. These images were manually interpreted to determine the dominant land use found within every quarter section (160 acres) in South Dakota. The categories classified included urban, rangeland, agriculture, forest, water, wetlands, and barren land.* This phase of the land use inventory was intended to produce an educational tool in the form of a map that displayed the complexity of land use across South Dakota. The total production time for this map was 24 weeks. The total cost for production and dissemination of the map was $4,920, or 6¢ per square mile.

The Phase I state land use map provided an overview of the current land use of South Dakota. Due to the crude level of detail, the map could not be utilized for detailed planning activities. However, it did stimulate much interest in the value of Landsat data for large area investigations. While most requests were from educational organizations interested in its academic value, several agencies did utilize the map for management purposes.

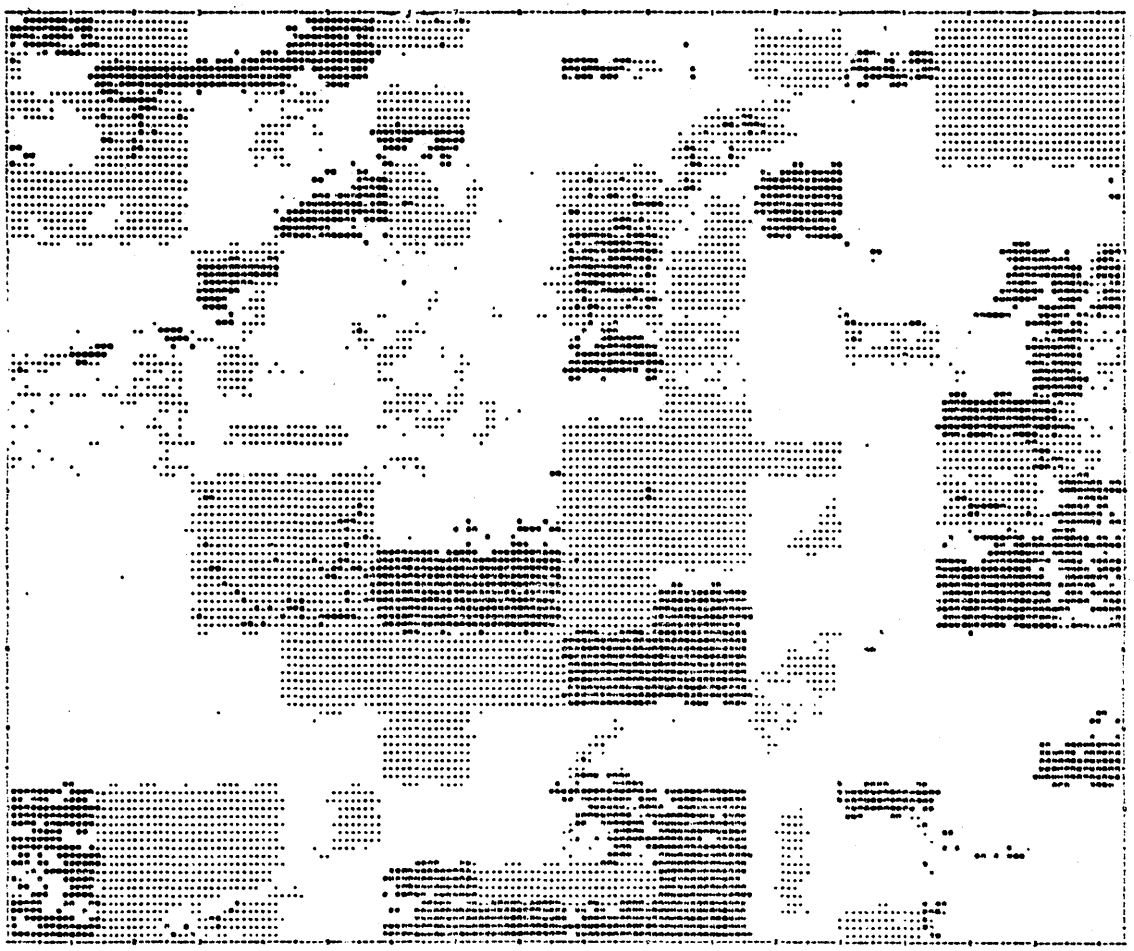
Phase II and III land use

State Planning Bureau anlaysts decided that more detailed land use information could only be obtained from Landsat data by utilizing more advanced analysis techniques than those used to produce the Phase I map. Landsat computer compatible tapes, which contain the same information as photographic Landsat imagery but in much greater detail (1.1 acre cells), provide this additional information. Computer software allowing this analysis was developed during fiscal years 1976 and 1977, and is contained in a computer package entitled *Landsat Imagery Analysis Package* (LIMAP).[4] These methods were employed early in Phase II to initiate a comprehensive, statewide land use inventory. The major differences between the output of this inventory and the Phase I land use mapping effort are:

(1) The maps produced in Phases II and III are based on data in 1.1 acre cells rather than 160 acre cells. The 1.1 acre data can be displayed with that cell size or aggregated to any meaningful larger cell size such as 10 or 40 acre cells (Figure 1).
(2) The classification scheme of Phases II and III contains more detailed land use categories. Table II contains a complete listing of the categories used in the digital land use inventory.
(3) Phases II and III maps are produced on a county

* This classification scheme is a Level I categorization adopted from: Anderson, J. R., et al. *A Land-Use and Land Cover Classification Scheme for use with Remote Sensor Data* USGS Professional Paper 964, Washington, D.C., 1976.

1.1 acre display



10 acre aggregation

Figure 1—Cow Creek Township, May 15, 1973, data aggregation

rather than on a statewide basis. The enormous quantity of data required to complete a 1.1 acre inventory of South Dakota necessitated storing the data in county files. These files, however, can be easily manipulated to produce maps of many different areas, including townships, watersheds, multicounty regions, or other regular and irregularly shaped areas.

(4) The digital format of the Phases II and III data permitted the development of computer software to rapidly scale the data to virtually any scale and cartographic projection. This added flexibility satisfies the needs of a variety of users by matching the scale of the land use data to customized base maps.

These increases in data content and display capabilities greatly enhance the potential applications of the land use data to planning and managing South Dakota's natural resources.

Current inventory status

The State Planning Bureau initiated the digital processing of Landsat imagery in 1975. As of September, 1977, 50 of 67 South Dakota counties had been mapped with four additional counties in progress.

The Planning Bureau has recently initiated a program to assess the accuracy of the completed portion of the land use inventory. Preliminary indications are that the average accuracy of the data is approximately 75 percent. Analysis results have progressively yielded higher accuracies with the most recently mapped counties being as accurate as 96 percent. Recognizing the importance of accurate land use data to decision makers, the State Planning Bureau has adopted a policy of only disseminating maps of at least 85 percent accuracy.

Costs

The time and cost requirements that must be met if the inventory is to be economically viable are critical. The State Planning Bureau has committed approximately five man-years and $300,000 toward the completion of the South Dakota Land Use Inventory. This includes a large amount of time and money devoted to computer software development. While this has increased both time and cost expenditures, it has allowed the formulation of an analysis system equipped to meet South Dakota's specific needs.

A detailed look at current analysis costs, excluding overhead, training and development expenses, shows the economic feasibility of recently produced land use maps. Analysts have been able to map a 3320 square mile area in northeastern South Dakota (approximately 4 counties) for a total analysis cost of $5,703.36. The Landsat data was mapped using four categories: water, large grains, small grains and pasture. The level of detail was 1.1 acres and the cost per square mile was $1.72 or $0.0027 per acre. Table III contains a detailed listing of mapping costs. These figures will not necessarily apply directly to other geographic areas, but can be considered fairly typical.

Applications

The true value of any land inventory is best determined by its applicability to management and policy problems. In addition to the South Dakota applications mentioned in Table V, the State Planning Bureau has provided publications and advice to 40 states and four foreign countries. While this indicates wide interest in the innovative nature of the South Dakota Land Use Inventory and the Land Resource Information System, it is the state applications that best illustrate the overall value of the program.

Limitations of Landsat data

While Landsat has generally been adequate as the primary data source for the land use inventory, it does have limita-

TABLE II—South Dakota Landsat Land Use Classification Scheme

Level I	Level II	Level III
100 Urban		
200 Agriculture		
	210 Cropland	
		211 Bare soil
		213 Small Grains
		215 Large Grains
300 Rangeland		
400 Forestland		
	410 Deciduous	
	420 Evergeeen	
500 Water		
600 Wetlands		
	630 Riparian Vegetation	
700 Barren Land		

TABLE III—Time and Cost Data for Digital Land Use Mapping of a 3320 Square Mile Area in Northeastern South Dakota.

TIME	
Data Rectification	2 hours
Classification	21 hours
Verification (Accuracy)	68 hours
Final Map Production	18 hours
	109 hours
COSTS	
Landsat Imagery	
1 CCT @ $200	$ 200.00
Computer Time	
7.182 hours @ $625.00/hr.	$4,488.75
Salary	
109 hours @ $6.94/hr.	$ 754.61
Ground Data Acquisition	
4 counties @ $40.00/county	$ 160.00
Misc. Supplies (Paper, tapes, etc.)	$ 100.00
	$5,703.36

adequate for the task of mapping certain land use categories. Wetlands, for instance, have an appearance similar to large grains on Landsat data obtained during the summer months. This often results in the misclassification of wetlands. While techniques exist to correct this problem, they increase both analysis time and cost. Sensor resolution limitations preclude the use of Landsat for meaningful classification of the state's urban areas. Again, alternative inventory methods such as aerial photography are available but at an increased analysis cost.

Another limitation of Landsat data is its timeliness. Raw data in the form of computer compatible tapes is currently available between three and four months after the satellite has recorded the scene. This data turnaround is often inadequate for ad hoc analysis since categorization of the data can require up to an additional month. Landsat can be a timely tool for monitoring natural disasters such as floods and droughts. However, to be of use, the data should be available within one to three weeks. When this data availability problem is solved, the utility of Landsat data to South Dakota will be greatly enhanced.

When the limitations of the Landsat data are recognized, satisfactory results can be easily and consistently obtained. The South Dakota State Planning Bureau has generally been satisfied with the results of its statewide digital land use inventory. The Bureau plans to complete the land use inventory during fiscal year 1978, and then to publish a state land use atlas containing the resultant data. Because of the successes achieved so far, the State Planning Bureau intends to continue using Landsat as an important tool in the planning and decision-making process.

IMPLICATIONS TO THE COMPUTER WORLD

The accelerating development of state digital image processing systems and natural resource information systems will provide many challenges for the computer world over the next decade. A great deal of development will need to be completed on both hardware and software.

Multivariate statistical analysis of image data requires a great deal of computing power. As future image data sets grow an order of magnitude in size and additional channels are added to future sensors, much faster processors will be required. Mass storage devices capable of storing hundreds of megabits of data with access times comparable to main storage need to be developed for specialized processing applications such as data rectification and image enhancement. Video image display devices will have to be upgraded to handle larger volumes of data. Remote data transmission devices will have to be accelerated as distributed image processing systems become more prevalent and transmission requirements are greatly increased. Color hard copy output devices will need to be improved to upgrade product quality and reduce production costs. Above all, a great deal of system's integration work will be required to effectively and efficiently pull together all the various hardware components required for operational image processing applications.

On the software side, a great deal of development can be expected. Generalized applications software packages will need to be prepared for a wide variety of mainframes and minicomputers as more and more organizations adopt remote sensing technology. Much custom software development will be required to tailor such packages to meet individual user needs. Interactive systems will become more prevalent as user sophistication increases.

Overall, development of image processing and information system capabilities provides the computer industry with an important opportunity to contribute to improving the quality of life and the wisdom of resource management in America and the world. While land and resource shortages have not yet reached crisis proportions, world population growth is bringing that day increasingly closer. New technologies alone will not solve these problems. Hopefully they will prove to be useful tools to the people who must.

TABLE IV—Typical Landsat Applications of the South Dakota Land Resource Information System

Comprehensive Planning
County Comprehensive Plans
U.S. Army Corps of Engineers Watershed Planning
State and Regional River Basin Planning
HUD 701 Land Use Planning
Environmental Impact Assessment
Energy development impact analysis
Transportation corridor selection
Energy transmission corridor analysis
Surface Water Inventory
Impoundment location
Flood plain delineation
Land Use Change Analysis
Agricultural conversion
Urbanization

ACKNOWLEDGMENTS

The author wishes to acknowledge that the body of this paper was compiled and edited from papers prepared by J. R. Reid and A. W. Johnson, North Dakota Regional Environmental Assessment Program, and J. Schlessinger and T. R. Loveland, South Dakota State Planning Bureau.

REFERENCES

1. Regional Environmental Assessment Program, Technical Task Force Reports, 1975, 195 pp.
2. Anderson, J. R., E. A. Hardy, J. T. Roach, and R. E. Witmer, "A Land Use and Land Cover Classification System for Use with Remote Sensor Data," *U.S. Geological Survey,* Professional Paper 964, 1976, 28 pp.
3. Rogers, R. H., L. E. Reed, N. F. Schmidt, and R. Schecter, "Computer Mapping of Landsat Data for Environmental Applications," *Proc. of Workshop for Environmental Applications for Multi-Spectral Imagery,* Fort Belvoir, VA., 1975, pp. 118-134.
4. Tessar, P. A., and J. E. Eidenshink, "The Landsat Imagery Analysis Package: Automated Land Use Classification and Multidimensional Geographic Analysis," *Proc. of the Second William E. Pecora Memorial Symposium,* Sioux Falls, S.D., 1976, forthcoming.

In perspective—Meeting the image processing challenge for remote sensing

by PHILIP H. SWAIN
Purdue University
West Lafayette, Indiana

INTRODUCTION

Although man has been observing the condition of his environment from aerial vantage points for many decades,[1] developments of the 1950s and 1960s could be pointed to as the origins of modern remote sensing technology.[2] The last twenty to thirty years have seen dramatic advancements in the design of sensor systems, particularly in the measurement of infrared energy, the advent of the digital computer, and progress in modeling some of the processes associated with human intelligence, notably pattern recognition. Within this decade we have developed the capability to marry these advancements with spaceborne systems which can provide us image data of the earth—and other planets as well—of unparalleled scope, scale, resolution and timeliness. The "dimensions" of the data, however, overwhelm our human capabilities to assess and assimilate it. The human visual system cannot deal effectively with the four or thirteen or more than twenty concurrent images of the same scene produced by modern multispectral imaging systems. Nor can we begin to cope with the volume of image data that has suddenly become available through the orbiting of such systems.

And thus the challenge: to develop effective computerized image processing techniques, either to reduce the flood of data to proportions we can handle in a useful way or to amplify our powers to manipulate the data and discern the useful information contained in it. How are we meeting this challenge? First, let us look a bit more closely at what is involved.

UNIQUE ASPECTS OF THE PROBLEM

In remote sensing, information about the earth and its atmosphere is conveyed to the sensor system through the spatial, spectral and temporal variations of the electromagnetic energy emanating from the scene.[2] Typically we organize the spatial variations into an image which can then provide a two-dimensional characterization of "things" in the scene in terms of their shape, size, orientation, and context.

To a limited extent, the spectral or wavelength-dependent variations in the scene can also be incorporated into an image through the dimension of color. However, these variations can be quantified to only a limited extent in this way because discernible "color" is a net effect which is uniquely definable by at most three fundamental components. Thus since our modern sensor systems can precisely measure more than three distinct spectral components we are bound to suffer loss of data (and usually, but not always, of information) if we attempt to represent these components in terms of color. An alternative is to represent the various spectral components as multiple images rendered in gray tones. Unfortunately such a representation is not only awkward but fails to display explicitly the subtle relationships between spectral components which may be essential for characterizing different ground covers.

Attempting visual display of temporal variations simply makes matters worse. Assuming that images of a given scene collected at different times can be precisely registered (the technology for accomplishing this has in fact been developed[3]), color renditions and multiple gray-tone images can again be used to convey temporal variations in the scene. But taking these variations together with the key spectral variations leads to a level of complexity guaranteed to overburden the human sensory system.

Thus, a characterizing feature of remote sensing data is its multi-image nature. Roughly speaking, a computer is as capable of dealing with four or thirteen or twenty dimensional data as it is with three dimensional data.[4] So it is natural to turn to the computer for assistance in making use of remote sensing data.

We have already mentioned the volume of remote sensing data that is available. Even early airborne systems (circa 1960s) typically produced "scenes" consisting of 10^6 multispectral pixels ("picture elements"). Today we often wish to process ten times that many pixels as a single unit. By the early 1980s, improvements in sensor resolution will increase the figure by another order of magnitude.[5]

Data volume is therefore another characterizing aspect of remote sensing imagery. We need computers to catalog as well as to analyze very rapidly the massive quantities of available data. But even with computers serving in these capacities, the situation is characteristically more demand-

ing in the case of remote sensing than in any other application of image processing which comes to mind.

Remote sensing is very much an applied technology. Historically its development has been and continues to be most rapid when a potential application provides a focus. As a result, image processing for remote sensing has characteristically been a multidisciplinary effort, resulting in a need to make scientists who may not be computer-oriented feel "at home" with the image processing machinery. This has been accomplished by paying serious attention to the man/machine and man/data interfaces, developing, for example, English-like command languages to be used for directing the image processing operations.[6,7]

Having sketched some characteristic aspects of image processing for remote sensing, we shall now turn to an overview of some of the efforts which have been directed toward meeting the challenge, what is happening today, and some apparent needs for the future. In doing so, it is helpful to think in terms of fairly distinct image processing activities: enhancement, preceding either visual or machine-implemented analysis; analysis, typically but not exclusively classification; and product formating, storage and retrieval.

WHERE HAVE WE BEEN?

Attention began to focus on the computer analysis of multispectral remote sensing data in the mid-1960s.[8] To limit the task to one of manageable proportions, it was decided to concentrate in the main on the spectral variations in the data, practically ignoring, for the time being, the spatial and temporal variations. This choice was largely motivated by the wealth of information that the spectral domain was assumed to contain and optimism with respect to how much effort would be required to extract a significant proportion of that information.

Much of the earliest multispectral image data was recorded in analog form, either photographically on film or electronically on magnetic tape. Once it had been decided to use digital computers to analyze the data, techniques and hardware were needed to convert this data from analog to digital form and store it on computer-compatible tape. But of course the data analyst then needed a facility for displaying the digital data and referencing specific locations in it. These combined needs gave great impetus to the development of image digitizing and digital display devices. In essence, the skills and theories which had been developed for the photographic darkroom had to be translated into a new digital technology.

Statistical pattern recognition was quickly adopted as a promising approach for analyzing the multivariate data associated with each pixel in the multispectral image.[9] One of the earliest applications was crop species mapping. Commensurate with the relative complexity of this task (some of the spectral differences among crops in the field are fairly subtle), maximum likelihood classification assuming multivariate Gaussian data distributions was often used to accomplish the analysis.[4] Interestingly, although many alternative classification methods have been investigated since those early days, the Gaussian maximum likelihood classifier has remained the most widely used. Some reasons which account for this include its ease of implementation, its moderate level of computational complexity and memory requirements, and its ability to represent moderately complex decision surfaces in the feature space through use of second order statistics (class covariance matrices as well as means).

The products of the data analysis were largely in two forms: classification "maps" in raster format analogous to the data input, usually printed on conventional computer lineprinters using distinct characters to represent the ground cover classes; and tabular summaries listing total area coverage by class and, for selected areas for which adequate ancillary data were available, an estimate of classification accuracy. These products were fine for illustrating the kinds of information which could be extracted from remote sensing data, but they would eventually be found to fall far short of the needs of the potential users. The map makers needed photo-quality images with tightly controlled geometric characteristics and somewhat less detail than the computer analyses generally produced; and in color. The policy makers needed results reported according to specialized categories not always identifiable by their spectral properties alone; and on the basis of oddly shaped political or jurisdictional regions. Once the need for these types of products was realized, vigorous work toward their generation met with success.[10]

Most of the remote sensing data processing capabilities were initially implemented on general purpose computers. Although ideal for research and development purposes, these implementations were clearly inadequate for the high-volume processing which would be called for when satellite-borne remote sensor systems were orbited in the early 1970s. A few research organizations and commercial establishments attempted to predict the image processing capabilities which would be needed and to construct special purpose systems to meet the anticipated need. Although they succeeded in demonstrating that special purpose systems could achieve dramatically improved throughput performance compared to general purpose versions they also proved that the remote sensing image processing technology was advancing at a great rate: their systems were obsolescent even before producing their first results.

WHERE ARE WE TODAY?

Present-day image processing technology has gone beyond the spectral domain alone, making significant strides into the spatial domain and some tentative steps into the temporal domain.[8]

A number of cosmetic enhancements are routinely applied to the digital multispectral data.[11–16] These include a wide range of false color display techniques, geometric adjustments to match existing map products, edge enhancement, resolution enhancement (reduction of blurring due to various sensor system characteristics), and other filtering operations.[17] Methods have been developed for precisely registering multiple images of the same scene to a common tem-

plate scene.[3,18,19] All of these advances have improved the interpretability of remote sensing image data and increased the potential utility of the final product to the user.

Methods for extracting information from remote sensing image data have developed to the point where they are testing the boundaries of what can be accomplished based on the spectral domain alone. With respect to some potential applications, the results achieved with data available from current sensor systems, of relatively limited spectral coverage and course spectral resolution, have fallen short of the results anticipated. This has had the unfortunate effect of discouraging segments of the potential user community and causing them and many interested layman to prematurely judge the limitations of the technology.

Texture and spatial contiguity are two geometric characteristics of the image data which have been demonstrated to be useful in the analysis process. Texture characterizes some ground covers of interest; the challenge has been to find effective measures of observable texture.[20-22] Detection of spatial contiguity permits scene segmentation and the delineation and classification of "objects" in the data.[23] When a large number of "objects" consist of many pixels, this leads to greater classification efficiency and accuracy. Statistical "sample classifiers" have been developed for this purpose.[23]

The products of the image processing operations today are a far cry from those available even a few years ago. Computer-driven precision film writers, ink-jet and electrostatic printers have made possible excellent quality map-like products at increasingly palatable prices. Capabilities for digitizing arbitrarily shaped polygonal boundaries have made it possible to report quantitative areal tabulations on a very flexible basis.[10] For applications where areal estimates are more feasible than "wall-to-wall" inventories, sophisticated statistical design and evaluation methods have been developed.[24,25]

Although they are exceedingly expensive, a few relatively powerful image processing systems are available for remote sensing data processing.[26] In the development of these systems, great attention has been paid to facilitating the role of the system operator who, in fact, is assumed to be a very capable data analyst. Thus an important element of the system is an interactive high-resolution digital display with a color CRT. A programmable array processor rather than a hard-wired special purpose processor may be used to implement the compute-bound steps in the processing. The processes for determining the appropriate parameters for various stages of the processing still involve the data analyst and often require considerable trial-and-error, however. As a result, the throughput of these specialized systems remains rather limited.

THE NEEDS OF THE FUTURE

We know from the performance of skilled image interpreters faced with visual analysis of available remote sensing data that there remains a wealth of information in the spatial and temporal domains as yet untapped by computer-implemented algorithms. Effectively characterizing and extracting this information constitute the challenge for the future.

A number of image enhancements appear feasible but have yet to be developed sufficiently for routine application. Examples include haze removal, scene-to-scene tonal normalization, and boundary detection and enhancement. The utility of the data and the task of the data analyst would both be improved by development of effective methods for coordinating ancillary data with the associated image data. This is not a trivial matter when one considers the spectrum of data types and formats that can be involved. It is not at all clear how to most effectively coordinate point, polygon, and line data with image data.

With precisely registered multitemporal remote sensing data promising to become more readily available, what means can be developed to effectively display the data and to enhance features of interest (often changes) to the image interpreter or the data analyst? What new problems will we have to deal with due simply to the gigantic proportions of the data sets? (Current Landsat scan lines consist of just over 3200 pixels, each of which is 4-dimensional. Projected systems will yield over 6000 6-dimensional pixels per scan line[5]).

Judging from the evolving image processing research, quite a range of new approaches for characterizing and extracting spatial information will be tested on remote sensing data and appropriately adapted. Texture, a complex visual phenomenon, is one of the features which seems quite obviously useful to human photointerpreters. Many very different approaches to the quantitative characterization of texture have been attempted and, in some instances involving remote sensing data, have been moderately successful. More progress in this area is needed.[21]

Where texture is a rather local spatial phenomenon, context might be thought of as a property bridging the local on one hand and the global on the other. Context is another phenomenon used effectively in manual photo-interpretation, since it can be quite helpful in characterizing a pixel or an object to take into consideration the nature of its neighbors. It has been demonstrated that the capacity to do this can be incorporated in pattern recognition algorithms for remote sensing, although the computational cost is substantial.[27]

Global relationships in images can be characterized using syntactic pattern recognition techniques.[28,29] This approach to remote sensing image analysis is very important because there are many instances in which rather general spatial relationships are key identifying characteristics. Rivers can be discriminated from lakes because they are "string-like" rather than "blob-like"; clouds can be discriminated from snow because clouds cast shadows. To use this approach, however, requires the abilities to infer the characterizing relationships from typical imagery and to capture them effectively in "pattern grammars". These are very difficult problems of image analysis in general and much remains to be done before practical applications of syntactic pattern recognition will be seen in remote sensing.

The decision processes needed for remote sensing data analysis can often be cast rather effectively as hierarchical

and/or sequential processes.[30] One can envision, for example, classifying a pixel into successively more specific categories by selectively using additional spectral bands or other features. The features to be used could be adaptively determined by the decisions made at each stage. This sort of hierarchical process can lead to both faster and more accurate classification. Still another classifier model can be used to efficiently process multitemporal image data from successive passes of the sensor over the scene.[31] In this case, likelihood computations are "cascaded" and the results of each stage of computation are made available to the next stage when later data is obtained.

Early in the history of the development of image processing for remote sensing, the goal of total automation of the process was established. It was assumed that only through total automation could the throughput of the processing system be made adequate to the data volume and demand for processing. Now, more than a decade later, that goal still remains the ideal, but it is widely recognized that it may be some time before we learn how to make computers perform certain complex tasks that humans perform rather well. Experience has demonstrated that, at least for the present, we cannot devise streamlined automated analysis algorithms employing available technology which can serve as a reasonable replacement for the data analyst who has training related to the application of interest and the computerized analysis tools as well. For the foreseeable future, then, we can expect to see continued research to improve the effectiveness of the human data analyst as part of the total image processing system. We have already noted the need to better coordinate the various forms of data the analyst must use (point, line and polygon data as well as the image data). More effective means of interacting with the total data bank are needed, utilizing, in all likelihood, interactive display facilities with graphics, image overlay, and color image capabilities. Similar remarks apply to interaction with intermediate and final processing results so that the analyst can function in a feedback loop to iteratively improve the results obtained.

Most beneficial use will be made of both the sensor data and the image processing facilities when a common set of these resources can be made available to a wide range of potential users. One way to accomplish this is to store fairly detailed analysis results in a flexible "earth resources information system" capable of being interrogated and providing a wide range of graphical and statistical information. This can be thought of as the "users' data base", containing more "refined" information developed by applying image processing to the "remote sensing data base". Some efforts have already been made along these lines,[32,33] but the needs of the user community are not yet being widely met.

Finally, the still rapidly evolving computer technology—microprocessors and parallel and pipeline computation—have important implications for the future of remote sensing image processing. The sheer volume of the data to be processed has already provided motivation for implementing relatively conventional analysis methods (multispectral clustering and maximum likelihood classification) on the ILLIAC IV parallel processor. Demonstration runs have resulted in computational speed-up factors of two to three orders of magnitude (between 10^2 and 10^3).[34] Yet this represents a rather "general purpose" implementation using obsolescent technology. The near future will see availability of very high speed microprogrammable processors which can be dynamically organized into parallel/pipeline configurations potentially capable of performing staggering numbers of computations per second.[35] It will be no small challenge to utilize such computational power effectively. Significantly, the prospect of having such power available is allowing us to think about image processing in new ways, to consider developing computational methods which were infeasible for practical applications when conventional serial implementations had to be assumed. The programability of these advanced systems will at once ensure their general applicability and make them available for further evolving the remote sensing image processing technology.

REFERENCES

1. Fischer, W. A., "History of Remote Sensing," in R. G. Reeves, ed., *Manual of Remote Sensing,* American Society of Photogrammetry, Falls Church, Va., 1975.
2. Landgrebe, D. A., "The Quantitative Approach: Concept and Rationale," in P. H. Swain and S. M. Davis, eds., *Remote Sensing: The Quantitative Approach,* McGraw-Hill International Book Co., New York, 1978.
3. Anuta, P. E., "Spatial Registration of Multispectral and Multitemporal Digital Imagery Using Fast Fourier Transform Techniques, *IEEE Trans. Geoscience Electronics,* Vol. GE-8, No. 4, pp. 353-368, 1970.
4. Swain, P. H., "Fundamentals of Pattern Recognition in Remote Sensing," in P. H. Swain and S. M. Davis, eds., *Remote Sensing: The Quantitative Approach,* McGraw-Hill International Book Co., New York, Germany, 1978.
5. Harnage, J. and D. Landgrebe, eds., "Landsat-D Thematic Mapper Technical Working Group Final Report," NASA Johnson Space Center, Houston, Texas, 1975.
6. Phillips, T. L., ed., *LARSYS Version 3 User's Manual,* Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana, 1973.
7. Swain, P. and D. Germann, "On the Application of Man-Machine Computing Systems to Problems of Remote Sensing," *Software Age,* Vol. 2, August 1968, pp. 13-20.
8. Landgrebe, D., "Computer-Based Remote Sensing Technology—A Look to the Future," *Remote Sensing of Environment,* Vol. 5, 1976, pp. 229-246.
9. Fu, K. S., D. A. Landgrebe, and T. L. Phillips, "Information Processing of Remotely Sensed Agricultural Data," *Proc. IEEE,* Vol. 57, April 1969, pp. 639-653.
10. Swain, P. H., "Advancements in Machine-Assisted Analysis of Multispectral Data for Land-Use Applications," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 77CH 1218-7 MPRSD, 1977, pp. 336-343.
11. Anuta, P. E., "Geometric Correction of ERTS-1 Digital Multispectral Scanner Data," *LARS Information Note 103073,* Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana, 1973.
12. Anuta, P. E., "Spline Function Approximation Techniques for Image Geometric Distortion Representation," *LARS Information Note 103174,* Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana, 1974.
13. Crane, R. B., "Preprocessing Techniques to Remove Atmospheric and Sensor Variability in Multispectral Scanner Data," *Proc. Seventh International Symposium on Remote Sensing of Environment,* Environmental Research Institute of Michigan, Ann Arbor, Michigan, 1971, pp. 1345-1355.

14. Emmert, R. A., and C. D. McGillem, "Multitemporal Geometric Distortion Correction Using the Affine Transformation," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 73 CHO 834-2 GE, 1973, pp. 1B-24 to 1B-32.
15. Higgins, J. L., and E. S. Deutsch, "The Effects of Picture Operations in the Fourier Domain and Vice Versa," in F. Shahroki, ed., *Remote Sensing of Earth Resources,* Vol. 1, University of Tennessee, Tullahoma, Tennessee, 1972, pp. 460-480.
16. Ulmer, D. E., "Processing and Enhancement of Landsat Imagery," *Proc. IEEE Midcon/77,* November 1977.
17. Lintz, J., Jr., and D. S. Simonett, eds., *Remote Sensing of Environment,* Addison-Wesley, Reading, Massachusetts, 1976.
18. Barnea, D. K., and H. F. Silverman, "A Class of Algorithms for Fast Digital Image Registration," *IEEE Trans. Computers,* Vol. C-21, No. 2, February 1972, pp. 179-186.
19. Lillestrand, R. L., "Techniques for Change Detection," *IEEE Trans. Computers,* Vol. C-21, No. 7, July 1972, pp. 654-660.
20. Haralick, R. M., K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," *IEEE Trans. Systems, Man, and Cybernetics,* Vol. SMC-3, November 1973, pp. 610-621.
21. Weszka, J. S., C. R. Dyer, and A. Rosenfeld, "A Comparative Study of Texture Measures and Terrain Classification," *IEEE Trans. Systems, Man and Cybernetics,* Vol. SMC-6, April 1976, pp. 269-285.
22. Carlton, S. G., and O. R. Mitchell, "Image Segmentation Using Texture and Gray Level," *Proc. IEEE Computer Society Conference on Pattern Recognition and Image Processing,* Rensselaer Polytechnic Institute, IEEE Cat. No. 77CH1208-9 C, 1977, pp. 387-391.
23. Kettig, R. L., and D. A. Landgrebe, "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects," *IEEE Trans. Geoscience Electronics,* Vol. GE-14, No. 1, January 1976, pp. 19-26.
24. Wigton, W. H., "Use of LANDSAT Technology by Statistical Reporting Service," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 76 CH 1103-1 MPRSD, 1976, pp. PB-6 to PB-10.
25. Bauer, M. E., M. M. Hixson, B. J. Davis, and J. B. Etheridge, "Crop Identification and Area Estimation by Computer-Aided Analysis of Landsat Data," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 77CH 1218-7 MPRSD, 1977, pp. 102-112.
26. Carter, V. P., F. Billingsley, and J. Lamar, *Summary Tables for Selected Digital Image Processing Systems,* Open File Report 77-414, U.S. Geological Survey, Reston, Va., May 1977.
27. Welch, J. R., and K. G. Salter, "A Context Algorithm for Pattern Recognition and Image Interpretation," *IEEE Trans. Systems, Man and Cybernetics,* Vol. SMC-1, January 1971, pp. 24-30.
28. Fu, K. S., *Syntactic Methods in Pattern Recognition,* Academic Press, 1974.
29. Brayer, J. M. and K. S. Fu, "Application of a Web Grammar Model to an Earth Resources Satellite Picture," *Proc. Third International Joint Conference on Pattern Recognition,* IEEE Cat. No. 76CH1140-3 C, 1976, pp. 405-410.
30. Swain, P. H. and H. Hauska, "The Decision Tree Classifier: Design and Potential," *IEEE Trans. Geoscience Electronics,* Vol. GE-15, July 1977, pp. 142-147.
31. Swain, P. H., "A Versatile Classifier Model for Multiobservational Analysis" (abstract only), *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 77CH 1218-7 MPRSD, 1977, p. 307.
32. Bryant, N. A., and A. L. Zobrist, "IBIS: A Geographic Information System Based on Digital Image Processing and Image Raster Datatype," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 76 CH 1103-1 MPRSD, 1976, pp. 1A-1 to 1A-7.
33. Hicks, J. E. and T. Hauger, "Managing Natural Resource Data: Minnesota Land Management Information System," Council of State Governments, Lexington, Ky., May 1977.
34. Ray, R. M., III, and H. F. Huddleston, "Illinois Crop-Acreage Estimation Experiment," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 76 CH 1103-1 MPRSD, 1976, pp. PB-14 to PB-21.
35. Allen, G. R., L. O. Bonrud, J. J. Cosgrove, and R. M. Stone, "The Design and Use of Special Purpose Processors for the Machine Processing of Remotely Sensed Data," *Proc. Symposium on Machine Processing of Remotely Sensed Data,* Purdue University, IEEE Cat. No. 73CHO 834-2 GE, 1973, pp. 1a-25 to 1a-42.

Programming hardware for remote sensing image analysis

by DAVID G. GOODENOUGH
Canada Centre for Remote Sensing
Ottawa, Ontario, Canada

INTRODUCTION

The costs of processing large numbers of images can be reduced substantially if use is made of minicomputers combined with special purpose hardware and array processors. The experience with systems of the Canada Centre for Remote Sensing (CCRS) is drawn upon to show how parallel processing of images can be achieved. Data formats are described which accommodate a wide variety of imaging sensors from scanners to radars. We present the software architecture for the movement and analysis in parallel of large images stored on disks. One such system at CCRS is able to perform a 4-feature, LANDSAT maximum likelihood classification of seven million pixels into 93 classes in 15 minutes and a two-dimensional Fourier transform of a 512 by 512 image in 12.0 seconds. The array processor which achieves these rates is discussed at length including its software and hardware components.

CCRS IMAGE SOURCES AND STRUCTURES

The Canada Centre for Remote Sensing (CCRS) receives image data from the LANDSAT-1 and -2 satellites of the United States and from sensors on four CCRS aircraft. The primary digital image sources for CCRS are listed in Table I. In the near future NASA will launch the LANDSAT-C and SEASAT-A satellites for which we will receive data of Canada. The sensor in most demand now is the four-channel LANDSAT multispectral scanner (MSS). The synthetic aperture radar (SAR) of SEASAT-A may allow effective all-weather sensing of ice, ocean, and land features which make this also an important source of data. The satellite image sources are characterized by wide area coverage, frequent coverage (when compared with an aircraft program) and high data rates. CCRS pursues projects involving airborne MSS and SAR systems. Each image source produces data with unique characteristics and errors. For CCRS remote sensing systems, the received data are passed through a Ground Data Handling Centre (GDHC) which organizes and corrects the data for the most serious errors and produces a computer compatible tape (CCT) for subsequent remote sensing image analysis. By remote sensing image analysis we mean the semi-automatic extraction of information from a data set, which may be composed of data from multiple sources, and the generation of a tabular, graphical, or digital representations of this information for later use by resource managers. The primary output products desired from the CCRS Image Analysis System (CIAS) are summarized in Table II. With the exception of item (4) in Table II, all other products can be produced at CCRS by a system involving a general purpose minicomputer and two special purpose hardware processors. The objective of this paper is to present for your consideration observations and conclusions related to the programming and analysis of remotely sensed data in systems with hardware processors.

The LANDSAT-1 and -2 multispectral scanners[1] generates images which are framed at CCRS to correspond approximately to a ground area of 185 km by 181 km. Each frame contains approximately 3240 picture samples (pixels) along each of 2286 scan lines. The MSS generates through 24 detectors four spectral bands, the first three (MSS 4, 5, 6) of which are logarithmically scaled and the fourth band (MSS 7) is linearly scaled. The raw spectral values are each quantized to 6 bits. The CCRS-GDHC produces corresponding CCTs containing radiometrically corrected, 8-bit quantized images. These LANDSAT CCTs follow a flexible, self-defining format[2] often referred to as the CCRS-JSC format. The basic form of this format for LANDSAT MSS CCTs is summarized in Table III. This CCT format is used by CCRS for all digital images and by other countries for LANDSAT image production.

A digitized image or picture of L lines, B bands, and P pixels is an integer array which can be represented by the real function, G, where,

$$G(L,B,P)=\sum_{l=1}^{L}\sum_{b=1}^{B}\sum_{p=1}^{P} f(l,b,p) \qquad (1)$$

and $f(l,b,p)$ is the intensity in band b for a pixel at position p on scan line 1. The order of summation in equation (1) indicates that the image is primarily interleaved by lines and secondarily interleaved by band. Such an image is usually said to be line interleaved. In remote sensing image processing, two other image forms are common, $G(L,P,B)$ with secondary interleaving by pixel (pixel interleaved), and $G(B,L,P)$ with primary interleaving by band (band interleaved). Different sensors produce different data structures which must often be reformatted into the structure most compatible with the given image processing system. Such

TABLE I.—Primary CCRS Digital Image Sources
All data are stored on 2400 ft., 1600 bpi computer compatible tapes.

(1) LANDSAT-1 and -2 Multispectral Scanners (MSS)
—4 registered spectral bands
—3240 pixels by 2286 lines=1 frame
—8 bits/pixel
(2) Airborne Daedalus MSS
—11 registered spectral bands
—728 pixels by 4500 lines
—8 bits/pixel
(3) ERIM Synthetic Aperture Radar
—4 unregistered bands, 2 frequencies, 2 polarizations
—4000 pixels by 1800 lines
—8 bits/pixel
(4) Aerial Photographs (Color or Black and White)
—scanned on a microdensitometer
—3 registered spectral bands
—4000 pixels by 2400 lines for 3 bands
—12000 pixels by 2400 lines for 1 band
—8 bits/pixel or 10 bits/pixel.

reformatting can be quite expensive because of the large data volumes involved.

An image contains spatial information, in addition to its spectral information, which should also be used in the subsequent data analysis. The balance in information content carried by the spatial properties of an image and its spectral properties varies from sensor to sensor, image to image, and application to application. For each combination of sensor and application, it is necessary to study this balance between spatial and spectral information. Spatial properties of a digital image are often characterized by computed spatial features which measure the differences or similarities between a pixel and its neighborhood. Such spatial features may be used as texture measures to be added to the dimensionality of the feature set describing the image or they may be used to generate edges. Edges can in turn be used to indicate homogeneous regions for which averaging of spectral features would be appropriate as a data compression aid.

TABLE II.—Primary CIAS Output Products

(1) Computer compatible tapes
—flexible, self-defining CCRS-JSC format
—up to 64 bands or features
—for input to:
(a) PDS color-write microdensitometer (high resolution)
(b) CCRS Electron Beam Image Recorder (EBIR) System (medium resolution)
(c) Color Strip Film Recorder (low resolution)
(d) Grid Structured Geographic Data Bases
(2) Approximate maps of class distributions
(a) —scaled for UTM maps from 1:50,000 to 1:1,000,000
—single class or multiple class representations
—latitude/longitude reference
(b) —photographs of CRT display.
(3) Tables
(a) measurements of class areas
(b) summaries of spectral and spatial statistics by class.
(4) Polygon Data Files
—generalizations of grid structured classifications for input into large, polygon structured data bases
—transferred by a unique tape format.

TABLE III.—CCRS-JSC Format for LANDSAT-2 MSS Images
There are five types of records in the basic 2400 ft., 1600 bpi CCT.

(1) *JSC Header Record*
—3060 bytes long
—contains information about the subsequent organization of the data on the CCT, e.g. the quantization, the processing used, the number of bands or features (up to 64), the number of pixels and lines, etc.
(2) *LANDSAT Header Record*
—1440 bytes long
—gives information specific to CCRS-GDHC processing such as orbit number, frame number, image date, special processing flags, frame center latitude and longitude, etc.
(3) *Geometric Transformation Record*
—2700 bytes long
—contains the geometric transformation parameters necessary to move from LANDSAT coordinates to the Universal Transverse Mercator (UTM) grid.
(4) *Radiometric Look-up Table Records*
—there are 4 radiometric look-up table records, one for each spectral band
—each record is 1620 bytes long
—in each record the radiometric look-up tables for the six detectors for that spectral band used to correct this CCT are given.
(5) *MSS Data Sets*
—a data set is made up of 4 records, each 3780 bytes long
—each record contains the record number, ancillary data, and video data for each spectral band
—the ancillary data includes GMT start scan time, latitude and longitude at the line center, satellite altitude and roll, pitch, yaw, and calibration wedge data.

Adding dimensionality to the data set through the addition of texture features increases the processing costs not only by increasing data volume into the analysis system, but also by increasing the costs of ground truth acquisition. The higher dimensionality of the feature set requires that more training samples be collected to maintain the same level of accuracy in the elements of the covariance matrix in order to achieve higher classification accuracies with additional features.[3] More training samples require the detailed examination of larger areas on the ground. In addition, in order to have confidence in the statistical estimates of classification error, test sites developed from ground truth should be increased in corresponding size to the increase in size of the training sites. Error estimates of classifications from training sites are overly optimistic. Similar error of classification estimates from test sites are realistic if the procedures described in a previous paper[4] are followed.

The image tape is transferred to disk in a CCRS file structure family known as UNIDSK.[5] This is a flexible, grid data base structure which can accommodate up to 64 features. Any form of interleaving is supported by this structure. The file contains a Master Header, Secondary Headers, a Video Header, and the Video Data Set. The Master Header, 1024 bytes long, identifies the file, its format, and the geographic location. The Secondary Headers are variable in number and length and contain information specific to the history of this file and pointers to related files. The Video Header contains the details of the format and preprocessing of the video data in the following Video Data Set. This family of formats also allows random access of pixel values, carries sufficient information to allow one to re-trace the steps by

which the file was created, and permits images to continue onto multiple disk packs. This UNIDSK structure is used for storage and analysis of all images at CCRS.

CIAS OVERVIEW

The CCRS Image Analysis System (CIAS)[6] consists of two minicomputer sub-systems, the IMAGE 100 sub-system and the PDS sub-system. The PDS sub-system includes a DEC PDP-11/40 minicomputer and a PDS flatbed color read/write precision microdensitometer. The PDS sub-system is connected to the IMAGE 100 sub-system via two paths, a dual port disk drive and a high speed (1 Mbit/sec) interprocessor communication link (DMC 11). The PDS sub-system is used either to digitize color aerial photographs and produce a UNIDSK file for subsequent analysis on the IMAGE 100 sub-system or to write such a file onto color film.

The IMAGE 100 sub-system, shown in Figure 1, is used for all image analysis. This sub-system is controlled by a DEC PDP-11/70 minicomputer with 256K 16-bit words of core memory. There are two additional processors in this system, a modified General Electric IMAGE 100 and a CCRS array processor called the Image Analysis Processor (IAP). There are three parallel paths for data transfers, the UNIBUS, the RH70/DWR70 data bus, and the IAP-RH11-DB1 data path. The IMAGE 100 is an interactive pipeline processing system[6,7] which supports the acquisition and non-parametric classification of 512 by 512 four-feature images. The pipeline processor permits real-time ratioing of features and 4 by 4 matrix multiplication to generate linear combinations of features.

Classification of the image is performed by comparing in hardware all image pixels with the signature distribution of the training area as represented by the distribution of 4-vectors or cells of unit volume. The frequencies of occurrences of 4-vectors may be thresholded to alter the resulting

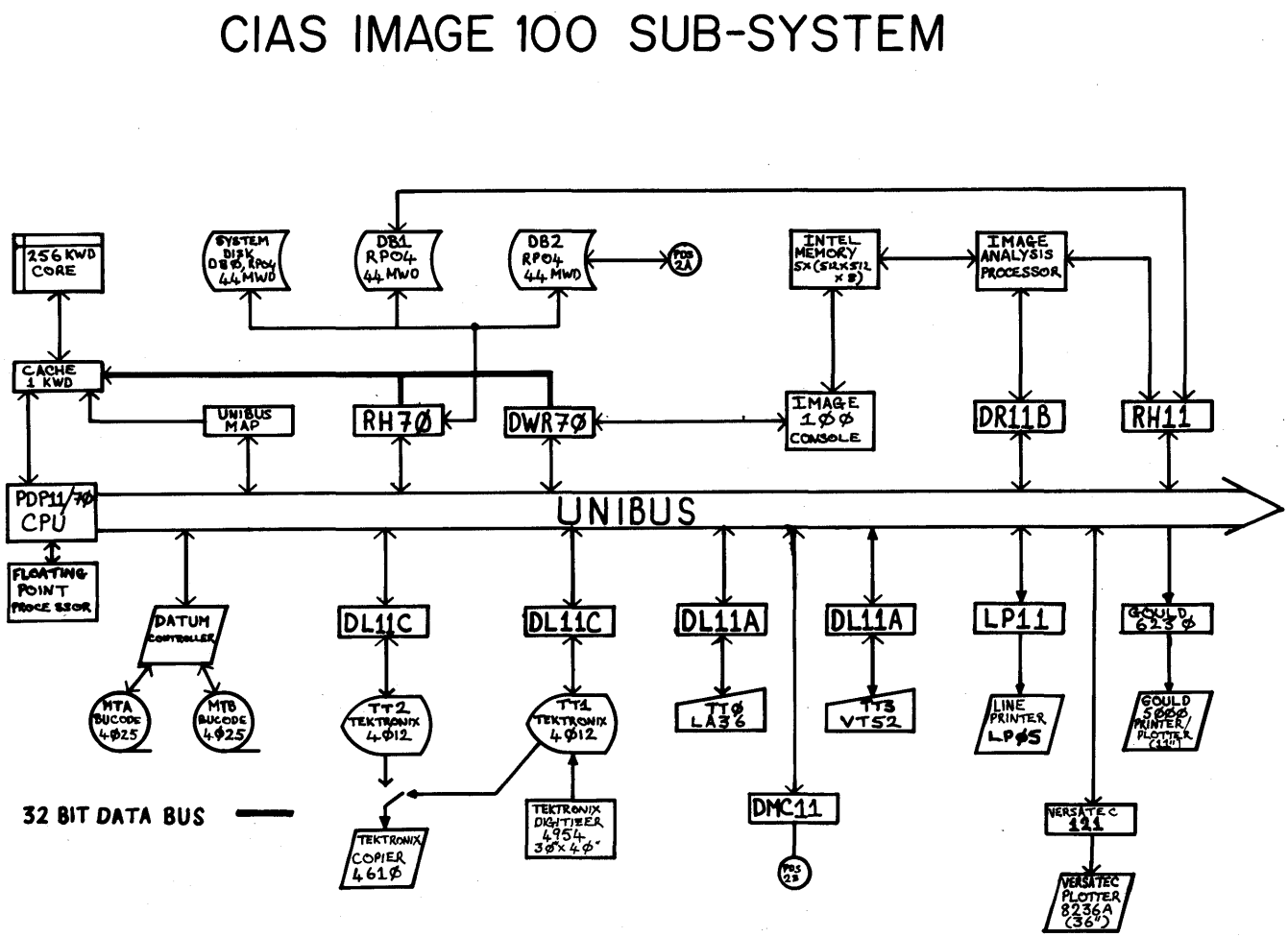


Figure 1—The IMAGE 100 sub-system of the CCRS Image Analysis System (CIAS) is shown. The sub-system is controlled by a DEC PDP-11/70 minicomputer. The other two processors in the sub-system are a modified General Electric IMAGE 100 and a CCRS/CDC array processor, the Image Analysis Processor (IAP). Note the parallel data paths permitting simultaneous image operations. The large disk, DB1, is dual ported to the IAP and serves as one of two image storage locations. As a result, the IAP can perform a maximum likelihood 93-class classification of LANDSAT four-feature frame in 15 minutes. Connections labelled PDS indicate links to the PDS sub-system of the CIAS.

classification. The resulting classification may be stored as a 1-bit overlay. The IMAGE 100 is very interactive and is particularly rapid for signature extraction. In supervised classification, a hierarchical approach is followed at CCRS from simple rectangular parallelepiped classification to non-parametric cell classification to maximum likelihood classification. The maximum likelihood classification may be performed in software in the PDP-11/70 or in hardware in the IAP. Unsupervised classification or clustering is performed in the following fashion. An image area is selected and the vector distribution determined for this area. Clustering by non-parametric or parametric methods of this distribution is carried out in the 11/70. The resulting class statistics can be loaded into the IAP and the classification executed on the whole image stored on the large disk labelled DB1 in Figure 1. Alternatively, the non-parametric class boundaries can be loaded into the IMAGE 100 and the 512 by 512 scene is then classified. Prior to the development of the IAP most of the system time was devoted to the maximum likelihood classification in the PDP-11/70 of large images.

The multiple processing capabilities of the CIAS are fully utilized under the DEC RSX-11D Version 6.2 multitasking operating system. The IMAGE 100 is controlled through an executive task. The software is written in DEC FORTRAN IV PLUS with the exception of a few input/output routines in MACRO-11. FORTRAN IV PLUS was selected because of the ease of transportability of such programs within CCRS and with our many user agencies and because of the efficient code generated by the FORTRAN IV PLUS compiler. For example, maximum likelihood classification executed three times as fast on an 11/70 in FORTRAN IV PLUS compared with DEC FORTRAN IV (FOR) on the same computer. The standardization of tape and disk formats for all our computers meant that software developed on our PDP-10/KI could be brought up in a day on our PDP-11/70. The CIAS interacts with the operator through program selection from a menu of programs presented on a Tektronix 4012 graphics terminal or by depressing any combination of function switches on the IMAGE 100. The menus available in the IMAGE 100 sub-system are listed in Table IV. Each menu is selected by letter. Each task listed in the menu is selected by number. A task in a menu may itself call a group of tasks. This approach can be contrasted with the use of complex, independent command driven tasks or picture processing languages. The menu approach is simple to implement, easy to maintain, and easily expanded. The operator is not required to learn a new language or symbol set, yet the tasks are grouped in a logical fashion to guide the operator in their use. In addition, the operator can always select directly tasks by name or by a menu letter-number combination. The two Application Menus are collections of tasks to support a particular remote sensing application such as forest inventorying, crop acreage estimation, and so forth.[6]

TABLE IV.—CIAS Image 100 Sub-System Menus

A.	Initialization/status
B.	Input/output
C.	Preprocessing
D.	Spatial Feature Generation
E.	Feature selection
F.	Signature acquisition
G.	Classification and clustering
H.	Classification filtering
I.	Utility
J.	Diagnostic
K.	Application menu #1
L.	Application menu #2

Image input to the IMAGE 100 sub-system is via the two 125 ips tape drives or the three disk drives. Training area selection, ground control points, and other map information may be input directly from the large digitizer tablet. Output maps and plots are generated on one of the two printer/plotters. Statistical summaries are produced on the line printer. The disk configuration permits parallel processing to be achieved and is discussed in the next section.

CIAS DISK CONFIGURATION

The arrangement of the three moving head disks in the CIAS is shown in Figure 2. Two of the large RP04 disks (88 megabytes) are dual ported. The disk unit DB0 is the system disk and has only one port. Parallel processing is achieved by connecting the Image Analysis Processor (IAP) to DB1 through the UNIBUS-B port of the RH11 disk controller. The PDP-11/70 through execution of programmed instructions directly controls all interfaces and thus, indirectly, all data transfers within the system. The core memory contains the programs under execution and the data buffers to be used by the Direct Memory Access (DMA) controllers, such as the RH70, RH11, and DR11B. Most data transfers between two devices in the CIAS use core memory as a transitory repository. Memory speed is thus a limiting factor on system data throughput with one exception, the IAP/DB1 link since it bypasses the core memory completely.

In Figure 2, the letters "D" and "C" are used to indicate paths along which data or control information, respectively, can flow. Control information between the PDP-11/70 and the three DMA controllers moves along UNIBUS-A. Data transfers from the DR11B and RH11 controllers to the 11/70 can also be carried out along UNIBUS-A. The impression of parallel transfers is created by the interleaving of these data transfers along UNIBUS-A to and from main memory. Eventually such transfers saturate the UNIBUS-A at a data rate of approximately 600 K words/sec.[8] The RH70 disk controller sends and receives control information along UNIBUS-A and Massbus-A, but directs data transfers along the Fastbus and Massbus-A. The RH11 disk controller communicates control information along UNIBUS-A but it may be selectively directed under program control to accept data transfers along either UNIBUS-A, UNIBUS-B, or Massbus-B. The two disk controllers and two data buses make it possible to have simultaneous data transfers to and from two disks.

Since the manufacturer of the dual port disks did not provide supporting dual port software, it was necessary for us to develop our own. The dual port disk DB1 can be either

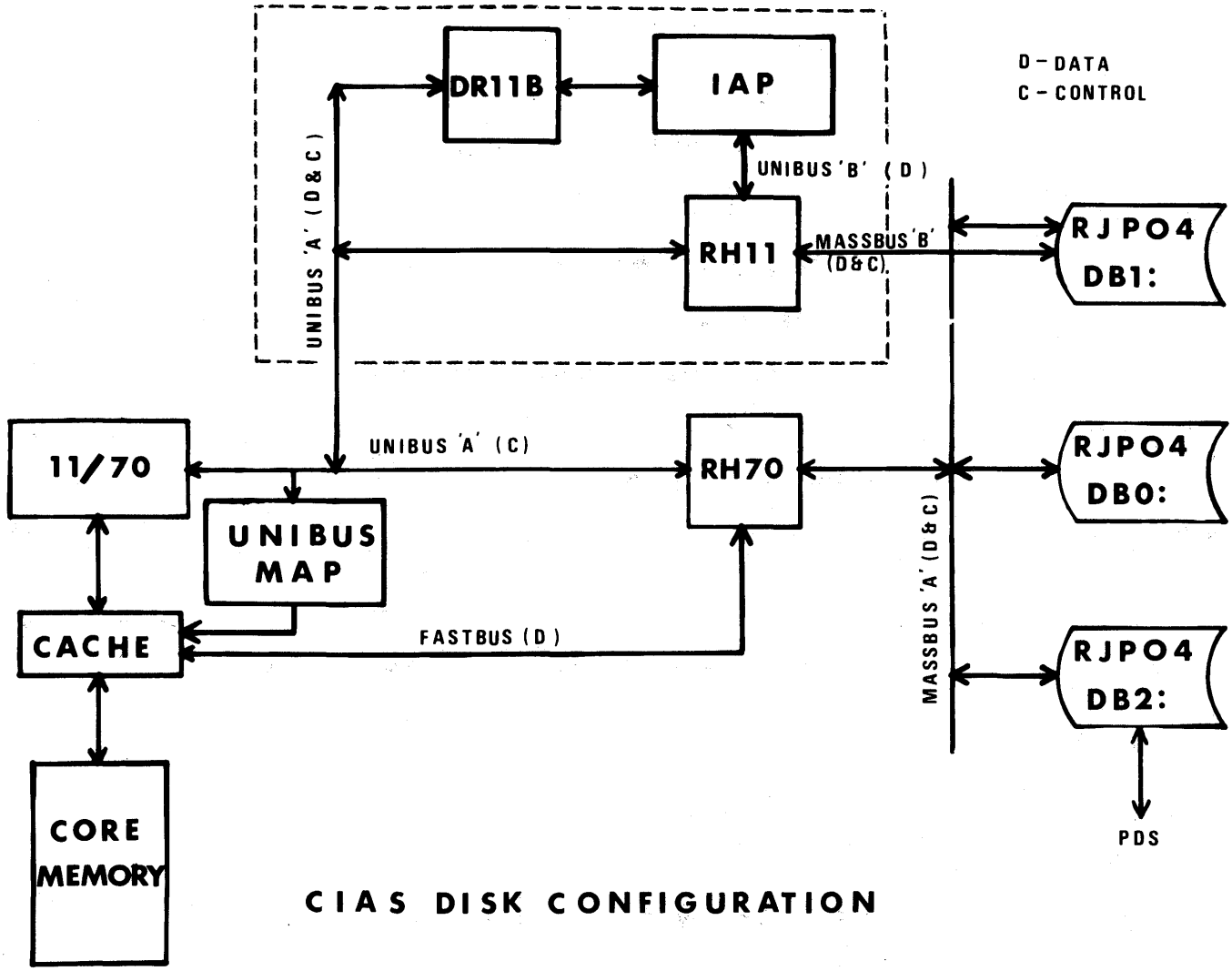


Figure 2—The details of the 88 megabyte disk connections for data (D) and control (C) information transfers are shown. The IAP can access DB1 while the 11/70 is simultaneously accessing DB0 or DB2. The IAP is also connected directly to the IMAGE 100. The RH11 and RH70 are DMA disk controllers. One disk handler handles all disk requests. Images are stored on disks in UNIDSK format (see text) compatible with DEC Files-11 structure. DB2 is used primarily for image transfers between the two sub-systems. Communication information concerning DB2 is passed along the interprocessor link, the DMC11, between two CPUs.

attached to the RH70 controller or the RH11 controller or unattached. In the unattached state either controller can attach the drive, at which point the other controller cannot access the drive. The dual port disk DB2 is interconnected between the PDP-11/70 and the 11/40 computers. This disk is used primarily for image transfers between the two subsystems and can, therefore, be dedicated to either computer for long periods of time. Communication information concerning DB2 is passed along the interprocessor link, the DMC11, between the two CPUs. For all images on disk, the UNIDSK format is used. This format has been implemented to generate files compatible with the DEC standard; namely, Files-11. The DEC disk handler, DB, passes file control functions to its ancillary processor task (F11ACP). Both DB and F11ACP have been modified to allow two additional functions which cause data transfer between the IAP and DB1. Dual port conflicts at DB1 are presented by forcing F11ACP to schedule sequentially all requests from a user task for DB1 data transfers. The IAP handler controls the DR11-B and RH11 controllers. The IAP handler does not directly accept from a user task requests for DB1 data transfers but receives such requests via F11ACP. The software development described here has enabled us to achieve the hardware data transfer rate limit of the RP04 of 2.5 μsec/word. The IAP is described in detail in the following section.

CIAS IMAGE ANALYSIS PROCESSOR (IAP)

CCRS, in cooperation with Computing Devices Company, have designed and implemented a special purpose array processor, the IAP. The IAP was intended to meet three

processing needs: (1) classification with the maximum likelihood decision rule; (2) computation of two-dimensional Fourier transforms; and (3) selection of pixels from disk images and the transferring of them to and from the IMAGE 100 solid state memory. An important cost consideration in the design of the IAP was to determine the number of bits to be carried in the arithmetic unit for the computations. The effects of coefficient quantization and arithmetic round off for the Fast Fourier Transform algorithm have been discussed previously.[10] The IAP was to calculate the FFT with decimation in time. The input data for the FFT were to be rearranged within the IAP to be in bit-reversed order prior to the pipeline arithmetic unit. The IAP would need to compute up to a two-dimensional forward and inverse FFT of a 512 by 512 input image and produce a result that agreed within 1 percent with that obtainable with 32-bit floating point arithmetic. Theoretical calculations for the FFT suggested that 18-bit floating point would likely be required. The IAP pipeline processor was then simulated on a PDP-11/40 for the FFT and the maximum likelihood calculations for four-feature images with 8-bit quantization.

The logarithm of the likelihood of observing a vector $\bar{x}$ given $\bar{x}$ is from class i with mean $\bar{m}_i$, covariance matrix C_i, and a priori probability P_i, is given by:[10]

$$l(\bar{x}/i)=\frac{n}{2}\ln 2\pi-\tfrac{1}{2}\ln \mid C_i \mid +\ln P_i$$

$$-\tfrac{1}{2}(\bar{x}-\bar{m}_i)^T C_i^{-1}(\bar{x}-\bar{m}_i)$$

The maximum likelihood decision rule is a comparison of this likelihood in which the vector $\bar{x}$ is assigned to the class for which the likelihood is a maximum and exceeds a selected threshold. The accuracy and speed of the computation is concerned chiefly with the last term, so that the preceding equation can be cast in the form for 4-feature data as:

$$L=\sum_{i=1}^{4}\sum_{j=1}^{4}(y_i Q_{ij} y_j)+K$$

where K is a constant, Q_{ij} is the ijth element of the inverse covariance matrix, and $y_n=x_n-m_n$. Because of the 16-bit minicomputer and the ease of 16-bit data manipulation, 16-bit fixed point with scaling for this equation was stimulated for the IAP pipeline processor with the number of bits for the Q-matrix and y values varied from 6 to 14 and 6 to 11 bits, respectively. The product was always truncated to a 16-bit integer. It was found that 16-bit fixed point with scaling was unsatisfactory and always produced results on real data for which the likelihood value differed by more than 10 percent from those obtained by a 32-bit floating point. With 18-bit floating point, the likelihood value was always within 0.1 percent of that obtained with a 32-bit floating point.

A simulation of the IAP pipeline processor for the FFT algorithm was then performed. The two-dimensional discrete Fourier transform, $X(k,l)$ is given by:[11]

$$X(k,l)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}x(m,n)W_M^{km}W_N^{ln}$$

with $k=0, 1, \ldots, M-1$ and $l=0, 1, \ldots, N-1$ and where $x(m,n)$ is the periodic input sequence, and $W_N=\exp(-j2\pi/N)$. The two-dimensional FFT (2-D FFT) was simulated by performing two forward FFTs followed by two inverse FFTs on sample 512 lines of 8-bit input data. Figure 3 is an example of the results obtained by this simulation. The ordinate axis is the root-mean-square variation of the absolute error in percent for the real image (RSMR). The absolute error in feature intensity in percent for the real image (APER)[8] when compared with the real image intensity resulting after a forward and inverse 2-D FFT was applied to the real input image is defined by:

$$\text{APER}=100\left|\frac{\text{Real Input}-\text{Real Result}}{\text{Real Input}}\right|$$

The maximum value of RSMR on the ordinate axis is 0.226 percent and the minimum value is 0.008 percent for 512 input 8-bit samples. The solid and dashed line curves correspond to having 5 bits or 6 bits for the exponents(E) of the floating point numbers, respectively, with a variable number of bits in the mantissa (M). It is clear from this figure that the error for E=5, M=13 is lower than that for

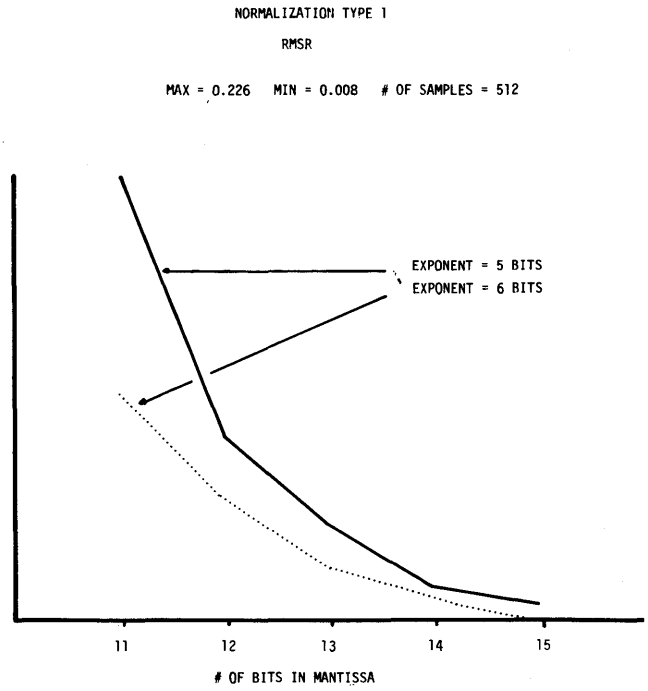


Figure 3—The IAP arithmetic pipeline was simulated for two-dimensional Fast Fourier Transforms (2D-FFT) by performing two forward FFTs followed by two inverse FFTs on sample 512 lines of 8-bit input data. The objective was to determine the number of bits required in the IAP computations to obtain the same result, to within 1 percent, as that obtained on a 32-bit machine. This figure is an example of the results obtained. The ordinate axis is the root-mean-square variation of the absolute error (RSMR) in percent for the real image after a 2D-FFT. The maximum value of RSMR here is 0.226 percent and the minimum value is 0.008 percent. For 18-bit floating point the real and imaginary image errors were always less than 1 percent. From this figure we see that 5 bits in the exponent and 13 bits in the mantissa gives the lowest error for an 18-bit floating point calculation.

E=6, M=12. A similar result was obtained for the imaginary images. We, therefore, selected 18-bit floating point with 5 bits in the exponent and 13 bits in the mantissa.

The interfaces of the IAP are indicated in Figure 4. The lettered lines connect to those in Figure 5 in which the IAP control computer and pipeline processor are shown. The IAP is interfaced to the PDP-11/70 for control and data through the DR11B. The IAP is interfaced to the RH11 so that the RP04 disk may be used for storage and retrieval of images in parallel with other CIAS activity. The IAP is also interfaced to the IMAGE 100 processor through the Line Buffer Memory (LBM). The LBM is a high speed random access memory (RAM) which can store 4096 bytes of image data at the 10 MHz data rate required to buffer the Intel 10 Megabit recirculating memory of the IMAGE 100 processor. The IAP can function either independently of the IMAGE 100 processor or in combination with it to use the ratioing and matrix multiplication hardware. The FIFO or First-In-First-Out memory receives data and control information from the DR11-B or data from the RH11. The FIFO is a 1000 18-bit word RAM which buffers the flow of data to permit the IAP's control computer to service higher priority activities.

The IAP Control Computer (CC) and Pipeline Processor (PLP) are shown in Figure 5. The programmable CC controls the initialization of registers which in turn control the other elements of the IAP. The CC performs any format conversion required of the data and is used to transfer data from one memory to another.[8] The instruction word for the IAP is 18-bits long. A cross-assembler between DEC Macro-11 style instructions and the IAP's microcode is used to simplify the programming of the CC. The Program Memory (PGM) shown in Figure 5 is used primarily to store the programs of the CC. The PGM consists[8] of 256 words of read-only-memory to store interrupt and bootstrap programs and 1 K 18-bit words of RAM.

The Pipeline Processor is an 18-bit floating point arithmetic unit composed of four major sections, the Pipeline Memory System (PLMS), the Scratch Pad Memory System (SPMS), the Pipeline (PL) and the Output Buffer. Once the PLP has been initialized by the CC, the PLP can perform independent computations. Thus, the CC and PLP can be operating in parallel. The PLP has a throughput rate of 100 ns which is achieved by having each section operate in parallel. The PLMS includes three 18-bit 1 K RAMs, marked as PLM1 to PLM3 in Figure 5. For maximum likelihood 4-feature, 8-class calculations, for example, PLM1 would contain 256 4-feature pixels, PLM2 would store 8 4-feature means, and PLM3 would have 8 inverse covariance matrices plus the decision rule threshold. Complex address calculators in the PLMS ensure that the address sequence required for FFTs is performed correctly. Data from the PLMS is called for by the SPMS which then delivers the data to the Pipeline according to the sequence of instructions held in the Scratch Pad (SP) memories. The SPs can hold up to 32 16-bit instructions plus 8 18-bit data words. Each SP program instruction is executed in 100 ns so that data may be passed to the Pipeline at that rate. The data from SP3 in Figure 5 is delayed in order to arrive at the second multiplier coincident with the output from the first multiplier. The output of the second multiplier is a 23-bit floating point number with 5 bits for the exponent and 18 bits for the mantissa. To achieve the required speed in the accumulator, it was found necessary to convert from a floating point number to a fixed point (47 bits) number (unfloat in Figure 5), accumulate, and then convert back to an 18-bit floating point number. When the required number of terms have been summed, the computed likelihood value, for example, is passed to the Output Buffer. The Output Buffer simplifies the programming of the PLP and determines the maximum likelihood value, thresholds this value, and outputs the resulting class number (16-bit integer) and likelihood to value to PLM1. The class number may be coded as an integer or with one bit set corresponding to the class. In the FFT mode the real and imaginary values are returned to PLM1 and PLM2, respectively. Thus, as data are flowing out of the PLMS, results are flowing into the PLMS. When the data have been processed, the array of results are sent to DB1, the IMAGE 100 or to the PDP-11/70 memory.

The size of memories in the PLP restrict for maximum likelihood the number of features and classes which may be accommodated in one pass. The number of terms, N_T, in the maximum likelihood expression for N features is: $N_T=N(N+1)/2$. If we assume one SP instruction is required for each term, then only 32 terms can be accommodated by the SP memory. If $N=7$ features, then $N_T=28$ terms which means the IAP is limited to 7 features in one maximum likelihood pass. The maximum number of classes, $M_{\max}$, for one pass is limited by N and by the size of the PLM according to: $M_{\max}\cdot(N_T+1)\leq 1024$. For four features, $M_{\max}=93$ classes. Other examples are listed in Table V. The PLP can calculate the maximum likelihood function for M classes and N features at a rate of $10^{-7}\cdot M\cdot(N_T+1)$ seconds per pixel. Program and input/output overheads increase this time for large images as is evident in Table V.

The IAP, although originally designed for rapid quadratic and FFT calculations, has other important features. The IAP can perform table look-up in 200 ns through one CC instruction. This is useful for radiometric and atmospheric corrections, image enhancements, and other image intensity manipulations. The IAP can reformat data simultaneously as they are moved from one memory to another which reduces substantially the processing costs associated with different input formats. The IMAGE 100 can display up to 8 classes as colored 1-bit overlays over a 512 by 512 pixel scene. The IAP can convert rapidly class numbers into 1-bit overlay patterns for subsequent IMAGE 100 display.

In classifying an image, the CIAS can first be used in a learn mode with sub-images displayed on the IMAGE 100. In this mode representative areas for supervised classification or for clustering are selected and parametric statistics for each class or cluster generated. The supervised classification is interactive and flexible. Parameters entered in response to teletype requests for a program are entered in files for subsequent utilization by that program and the IAP in a bulk processing mode.[8] For the classification of an N-feature, 512 by 512 scene, the statistics and input video files are read by the IAP and the scene classified. Figure 6 shows

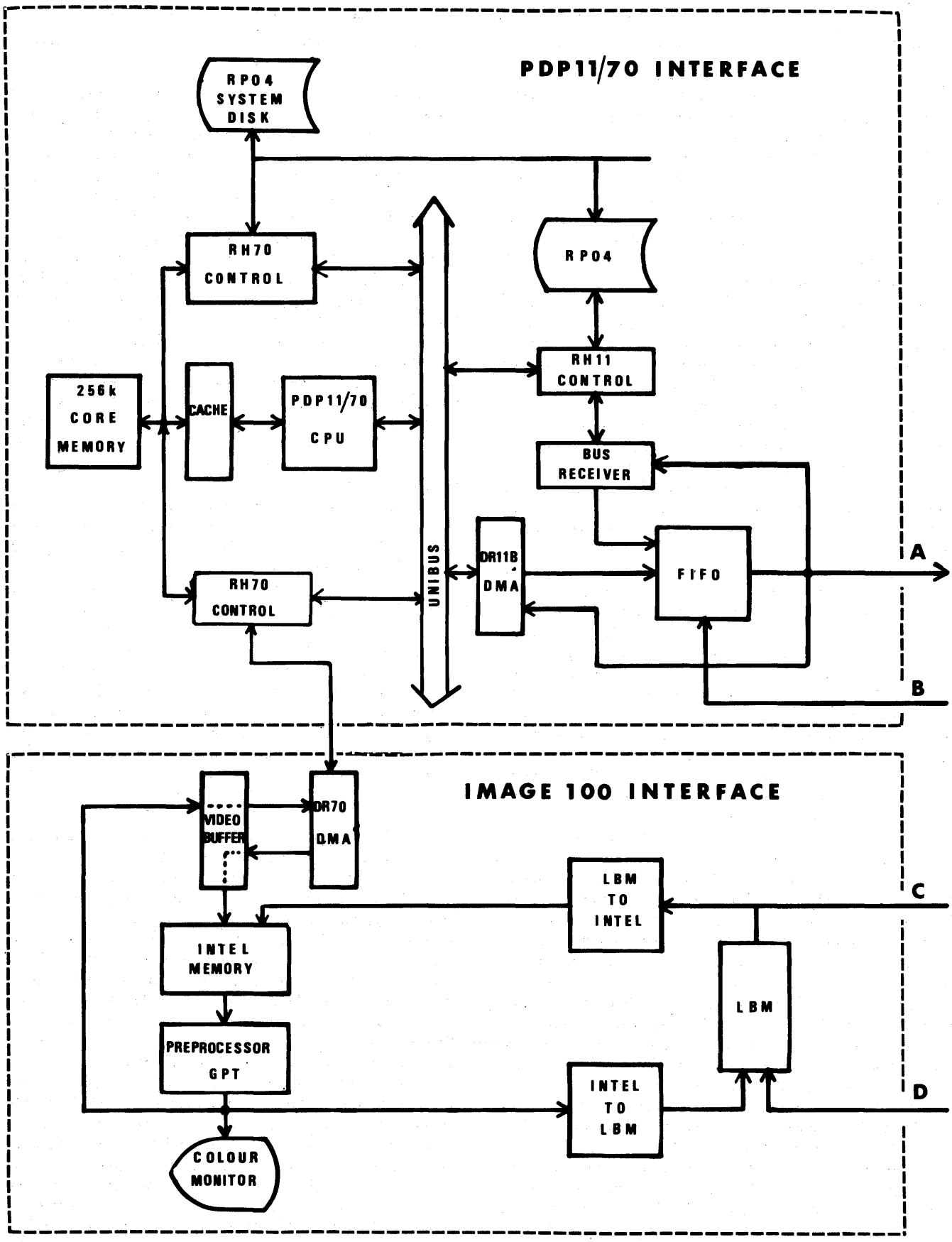


Figure 4—The IAP is interfaced through the First-In-First-Out (FIFO) 1 K word memory to the PDP-11/70 through the DR11B and to a large disk through the RH11 disk controller. The IAP is also interfaced to the IMAGE 100 through the 4 K bytes of Line Buffer Memory (LBM). This latter connection allows the IAP to use the ratioing and matrix multiplication hardware of the IMAGE 100, to use the 10 megabit Intel memory, and permits rapid image transfers between the IMAGE 100 and the disk DB1 through the IAP. The lettered lines connect to Figure 5.

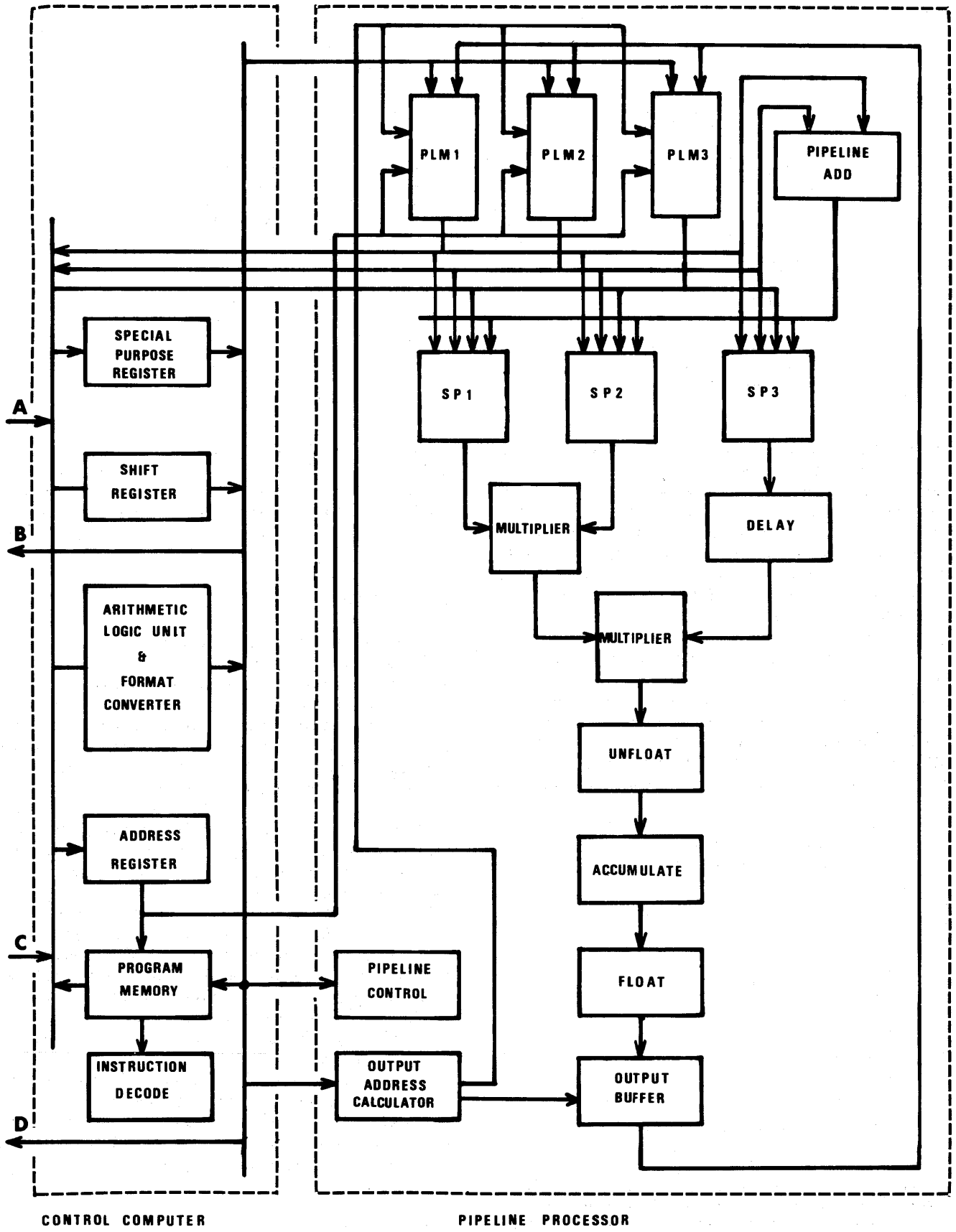


Figure 5—The IAP's Control Computer (CC) and Pipeline Processor (PLP) are shown. The CC is programmable and controls the initialization of registers which in turn control the other elements of the IAP. PLMs are 18-bit 1 K RAMs which hold the data, means, and covariances for maximum likelihood, for example, and in which the results from the output buffer are stored. Scratch Pad (SP) memories hold instructions on which PLM to select and deliver data to the pipeline. The pipeline computes quadratic summations at 100 ns per term. The IAP can perform a forward 2D-FFT of a 512 by 512 8-bit input image in 12 seconds. It can also perform a table look-up in 200 ns and reformat data rapidly as they are moved from one memory to another.

TABLE V.—CIAS/IAP Processing Times

I. Maximum Likelihood Classification: (excludes statistics generation)	No. of Classes	No. of Channels	Line Length	No. of Lines	Processing Times
LANDSAT-1,2	32	4	512	512	12 sec.
	32	4	3300	2286	5.8 min.
	93(max)	4	3300	2286	15.0 min.
LANDSAT-3	64(max)	5	3300	2286	15.2 min.
MSS(1 Pass)	35(max)	7	2048	2048	8.2 min.
Multisensor or Multitemporal (3 Pass)	32	11	2048	2048	20 min.
II. 2-D-FFT		1	512	512	12.8 sec. forward 17.0 sec. inverse
III. IMAGE 100 Transfers from Disk					
Decimate a LANDSAT Frame to:		4	512	512	28.0 sec.
Move an I-100 Screen to:		4	512	512	12.0 sec.

how the processing times for the maximum likelihood decision rule (MLDR) vary with number of features (4 to 7) and number of classes. The stars in Figure 6 mark the maximum number of classes which can be processed in one pass for the number of features indicated. At low numbers of classes one is limited by the minimum time (2.5 seconds) required to input the image to the IAP. The classification results flow to core memory and then to disk. It is of interest to note that for more than 21 classes, an image on a 1600 bpi magnetic tape can be directly read in on our 125 ips drives and classified by MLDR with no loss in processing speed.

Table V summarizes the processing rates achieved by the CIAS with the IAP for maximum likelihood classification, 2-D Fast Fourier Transforms, and transfers from the large DB1 disk to the IMAGE 100 solid state memory. These rates are high enough that most of the CCRS bulk processing

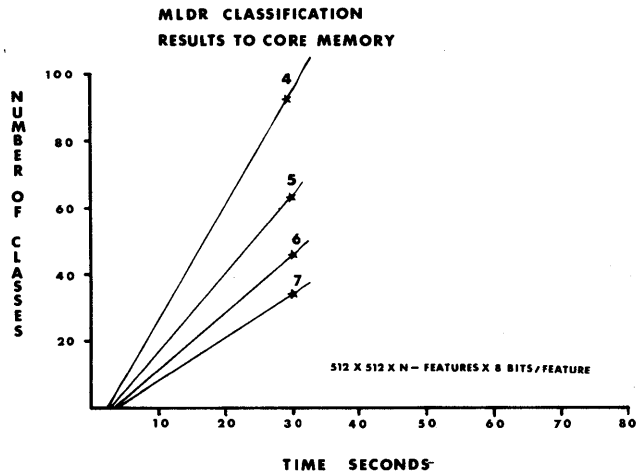


Figure 6—The variation of IAP processing times are shown for maximum likelihood decision rule (MLDR) classification for normal distributions for a 512 by 512 image of N features with 8 bits per feature. The number of features has been changed from 4 to 7. The stars mark the maximum number of classes the IAP can process for the indicated feature space dimensionality. At low numbers of classes one is limited by the minimum time (2.5 seconds) required to input the image to the IAP. The classification results flow to core memory and then to disk.

requirements for LANDSAT data can now be met by the CIAS. The parallel processing permitted in the CIAS by the 11/70, IMAGE 100, and IAP and the parallel processing within the IAP itself mean that the rates (within 10 percent) given in Table V are achieved even under full system load.

CONCLUSION

The minicomputer when combined with an array processor and display hardware can match or exceed the image processing performance of larger, general purpose computers. The complexities caused by the additional hardware can be reduced by following programming standards and conventions consistent with the software operating environment selected for the system. The use of a single standard tape format and a single disk format minimizes the number of input/output programs required to support the additional hardware. The inclusion of parallel processing and multitasking permits a minicomputer based system to achieve phenomenal image processing throughput at lower costs than conventional systems.

Flexibility can be maintained without inordinate sacrifices in speed. It is often expressed that software image processing systems are much easier to change than systems with special processors. Our experience has been that software systems of hundreds of programs developed a rigidity comparable to systems with hardware processors. Both approaches require that good documentation and change procedures be followed.

The application focus of this paper has been image processing in remote sensing. Many of the techniques presented here are, however, applicable to other areas of image processing and can be used in those areas to obtain lower cost solutions to their image processing problems.

REFERENCES

1. *Earth Resources Technology Satellite Data User's Handbook,* Canada Centre for Remote Sensing, Ottawa, Canada.
2. Strome, W. M., S. S. Vishnubhatla and F. E. Guertin, "Format Speci-

fications for Canadian LANDSAT MSS System Corrected Computer Compatible Tape," CCRS Report 75-3, 1975, Ottawa, Canada.
3. Boullion, T. L., P. L. Odell and B. S. Duran, "Estimating the Probability of Misclassification and Variate Selection," *Pat. Rec.* 7, 1975, pp. 139-146.
4. Goldberg, M., D. G. Goodenough and S. Shlien, "Classification Methods and Error Estimation for Multispectral Scanner Data," *Proc. 3rd. Cdn. Symp. on Rem. Sens.*, 1975, pp. 125-143, Edmonton, Alta.
5. Goodenough, D. G., K. J. O'Neill, J. Prinz and A. Smith, "CCRS Universal Image Disk Family of Formats (UNIDSK) for PDP-10 and PDP-11 Computers," CCRS Report, 1978, (in preparation).
6. Goodenough, D. G., "The Canada Centre for Remote Sensing's Image Analysis System (CIAS)," *Proc. 4th Cdn. Symp. on Rem. Sens.*, Quebec, Canada, 1977, pp. 227-245.
7. Goodenough, D. G., "IMAGE 100 Classification Methods for ERTS Scanner Data," *Cdn. J. Rem. Sens.*, 2, 1976, pp. 18-29.
8. Goodenough, D. G., S. F. Gourley, K. J. O'Neill and A. Wilson, "Bulk Image Processing at CCRS," presented at the SPSE/SPIE 1977 Tokyo Symposium on Photo and Electro Imaging, Tokyo, Japan, 1977.
9. Oppenheim, A. V. and R. W. Schafer, *Digital Signal Processing*, p. 447 ff, Prentice-Hall: New Jersey, 1975.
10. Goodenough, D. G. and S. Shlien, "Automatic Classification Methodology," CCRS Report 74-1, CCRS, Ottawa, Canada.
11. Oppenheim, A. V. and R. W. Schafer, *op. cit.*, p. 290 ff.

Use of textural features in the analysis of Landsat images

by H. K. RAMAPRIYAN, S. H. CHANG, and R. L. McKINNEY

Computer Sciences Corporation
Falls Church, Virginia

INTRODUCTION

The use of remotely sensed data for land-cover categorization has been well demonstrated in recent years.[1] In addition to the use of conventional methods, such as windshield surveys and low- and high-altitude aerial photographs, satellite data are being employed by several planning agencies for creating land-use and land-cover data bases to assist in decisionmaking.

Human interpretation of aircraft and satellite imagery is still used extensively and is essential for certain applications (e.g., lineament studies in geology). However, automatic interpretation of the image data using digital computers significantly complements human analysis and is advantageous in providing consistency, labor savings, and timely availability of interpreted data for postprocessing and inclusion into data bases.

The major differences between human and machine interpretation of images are as follows. A human interpreter uses color, spatial variation of color, and relative location of objects (i.e., spectral, textural, and contextual features) in an image simultaneously. Starting with a synoptic view of the image, the human interpreter partitions that image into homogeneous segments and associates labels with each of them. It is difficult to design algorithms for machine interpretation which simulate the human photointerpreter. However, machine analysis has the advantage of high resolution not achievable with human interpretation. Although it is possible to incorporate spectral, textural, and contextual features into machine analysis, the most commonly used classification algorithms employ only the spectral data and assign categories (or class labels) to single resolution elements (pixels) independently of other pixels.

The accuracy of classification maps is a principal concern for users. Studies of classification algorithms have been conducted which compare ground-truth maps with the classification maps generated by machine interpretation of Landsat multispectral data using only the spectral features at individual pixel locations. For example, in Reference 2, confusion is shown to exist between certain urban and agricultural cover types. This is due partially to the similarity in the spectral characteristics and partially to the differences in human and computer characterization of classes (e.g., when a region with large residential lots is encountered, human interpretation for the entire region is "urban," whereas machine interpretation is dependent on the urban-vegetation mixture). To distinguish between categories exhibiting similar spectral reflectances, the use of additional features dependent on texture and/or context is helpful. For example, although the spectral characteristics of some urban and agricultural cover types are similar (due to the variety of objects constituting urban pixels and the relatively homogeneous nature of agricultural regions), the variations of spectral reflectances within an urban region can be expected to be greater.

In the past several experiments have been performed on the use of textural features in classification. A review of the work on textural feature definition is given in Reference 2. In Reference 3, a comparative study of texture measures used for terrain classification is presented; the features are categorized as based on Fourier power spectra, second-order grey-level statistics, grey-level differences statistics, and grey-level run-length statistics. Examples of textural features based on Fourier transforms of image segments are given in References 3-5. In Reference 4 textural features invariant under translation and rotation are derived in terms of discrete Fourier transforms of square image segments and are shown to improve the accuracy of classification. Several studies by Haralick and his coinvestigators[6-12] employ various textural features defined in terms of grey-level co-occurrence matrices which can be regarded as second-order grey-level statistics; "contrast," "angular second moment," "entropy," and "correlation," which are functions of such matrices, are demonstrated as particularly useful. Grey-level difference statistics, defined in Reference 3, are obtained in terms of the frequencies of occurrences of particular values of grey-level differences at given separations and orientations in an image segment. The features derived from the grey-level co-occurrence matrices and the grey-level difference statistics are somewhat similar. Other measures of texture are defined in Reference 13 in terms of the frequencies of "runs" of grey-levels in given ranges in specified directions.

The utility of these textural features has been demonstrated for terrain classifications using aerial photography and Landsat data. However, in all of these studies, the features have been defined over large blocks of data, which implies a loss of effective resolution. The purpose of this paper is to develop a set of features which are a compromise between using large blocks of data (reducing resolution) and

using only spectral information at a single resolution cell. These features will be defined at every point in the image but will use combinations of spectral reflectances at neighboring pixels of each pixel. Thus, an augmentation of the conventional spectral features with spatial and, in turn, textural information results. Preliminary experimental results on the reduction in dimensionality of such feature vectors and a comparison of classifications generated by the conventional and the proposed techniques are presented in the following sections.

STATEMENT OF THE PROBLEM

The notation to be used in the remainder of this paper is as follows. Let P denote the array of spectral measurements, with p_{ijk} denoting the kth spectral feature measured at the ijth pixel, where i equals 1, 2, . . . , M; j equals 1, 2, . . . , N; and k equals 1, 2, . . . , K (in the case of Landsat, K equals 4).

Conventional classification techniques convert P to an array, Q, called the classification map, where q_{ij} are numbers in the range 1 to L, i and j have the same range as in the case of P, and L equals the number of classes. The decision on the assignment of p_{ijk} to q_{ij} depends only on the K numbers corresponding to the ijth pixel in P.

The problems considered in this paper are (1) how to define features at every point in the image such that the original resolution of P is preserved while, at the same time, spatial dependences of spectral reflectances are utilized in making the class assignments in Q and (2) how to use these features to obtain improved classifications without significant increases in computation.

FEATURE DEFINITION

In terms of the notation introduced previously, simple textural augmentation of spectral features can be provided as follows. For each ij define a feature vector, x_{ij}, given by

$$x_{ij}=\{p_{ij} \mid p_{i,j+1} \mid p_{i+1,j} \mid p_{i+1,j+1}\} \quad (1)$$

where p_{ij} represents the K-dimensional feature vector at ij (assuming all vectors to be row vectors).

Thus, with Landsat data 16-dimensional feature vectors result. Other similar feature vectors can be defined at every point in the image. Examples are the following:

$$x_{ij}=\{p_{ij} \mid p_{ij}-p_{i,j+1}\mid \mid p_{ij}-p_{i+1,j}\mid \mid p_{ij}-p_{i+1,j+1}\mid\} \quad (2)$$

$$x_{ij}=\{p_{ij} \quad p_{i-1,j} \quad p_{i,j-1} \quad p_{i+1,j} \quad p_{i,j+1}\} \quad (3)$$

$$x_{ij}=\{p_{ij} \quad \min_{k,l=\pm 1} \quad p_{ij}-p_{i+k,j+1}\mid \max_{k,l=\pm 1} \mid p_{ij}-p_{i+k,j+1}\} \quad (4)$$

(In all of the foregoing equations, special definitions must be made for the cases in which the subscripts of p exceed their ranges. A reasonable definition is to replace $p_{i+1,j}$ by $p_{i,j}$ and so on.)

Each of these feature vectors includes the spectral features at the point ij as well as those in the immediate surroundings of ij. These vectors could simply be used in a conventional classification algorithm such as a maximum-likelihood supervised classifier. However, because the dimensionality of the feature space has effectively been increased by a factor of four (five and three in the case of Equations (3) and (4), respectively), it is desirable to reduce the dimensionality by some feature selection or extraction technique and then perform the classification.

In the experiments discussed here, the feature vector definition (Equation (1)) is used. The Karhounen-Loeve (K-L) principal components transformation[14] is used to reduce the dimensionality. This linear transformation is derived from the orthonormal eigenvectors of the covariance matrix, C, of the 4K-dimensional data. By arranging the eigenvectors such that they correspond to the eigenvalues of C in descending order, the first few components of the transformed feature vector can be used for classification, because most of the significant information is contained therein. An alternative method is a canonical transformation,[15] the use of which facilitates separability of the categories of interest. This transformation is derived using the statistics of the individual classes to be separated. The transformed components are such that the separations among classes are in descending order from the first to the last component. The first few components can therefore be chosen for classification. The K-L transformation was used here because the software implementing it was readily available.

EXPERIMENTAL RESULTS

A 250-by-250-pixel subset of a Landsat scene (ID 1999-15091) covering Orlando, Florida, was used in this experiment. The four bands of the image are shown in Figure 1. Using the feature definition of Equation (1) and the K-L transformation, textural features (the features including the spatial and spectral information) were extracted. The first eight of the individual components so derived are shown in Figure 2. The first two components are effectively low-pass filter output, whereas the subsequent components tend to emphasize the edges of the image. This is also evidenced by an examination of the eigenvectors used for the transformation as shown in Figure 3.

Using training samples, eight classes were defined in this scene:

Class	Name	Description
1	Residential 1	Residential with significant tree cover
2	Residential 2	Residential with sparse tree cover
3	Vegetation 1	Dense trees and shrubs
4	Vegetation 2	Dense trees
5	Vegetation 3	Dense shrubs
6	Vegetation 4	Sparse trees and shrubs
7	Water	Water
8	Swamp	Mixture of wetlands and vegetation

The image was classified using the spectral and textural

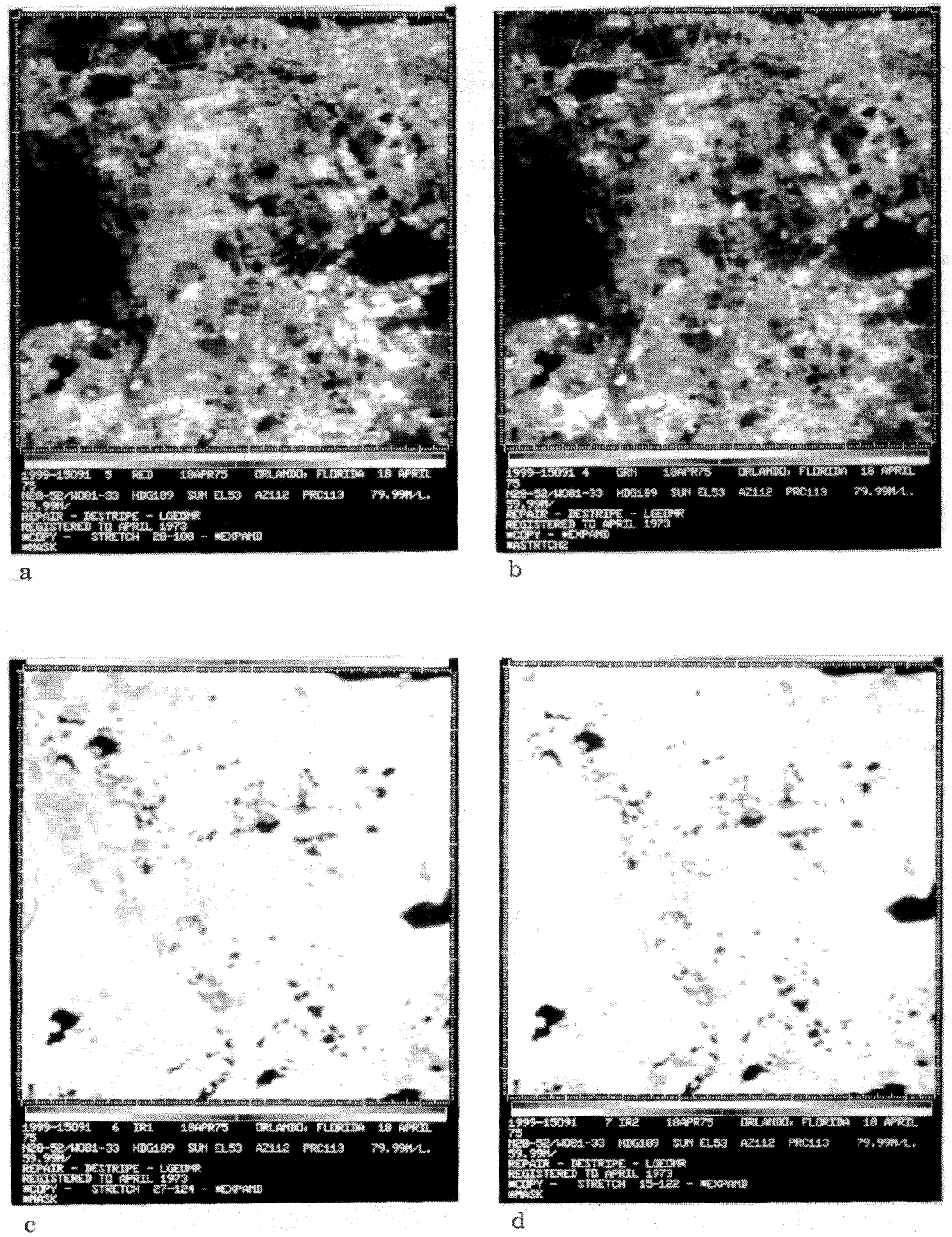
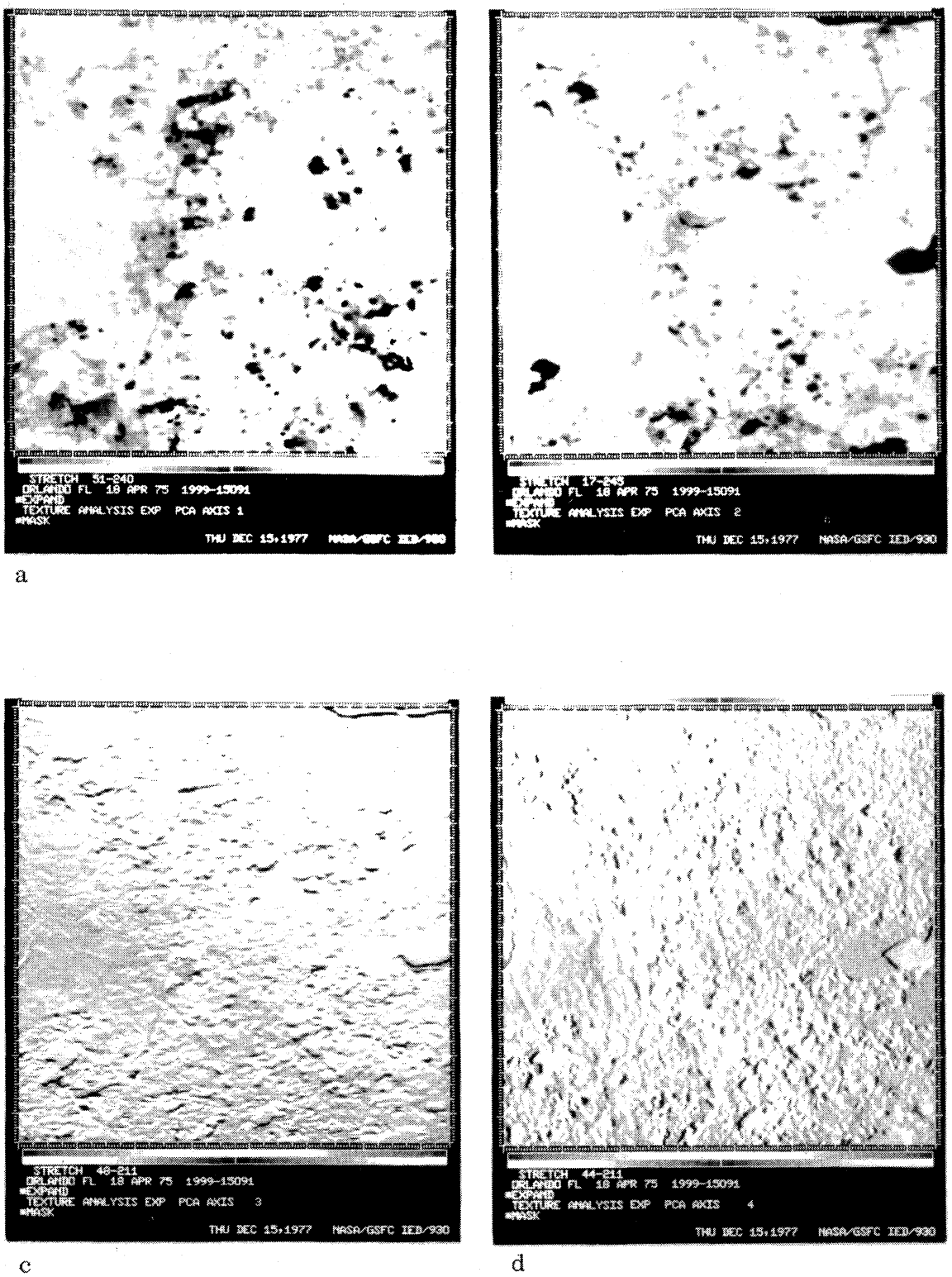


Figure 1—A subset of the Landsat scene covering Orlando, Florida

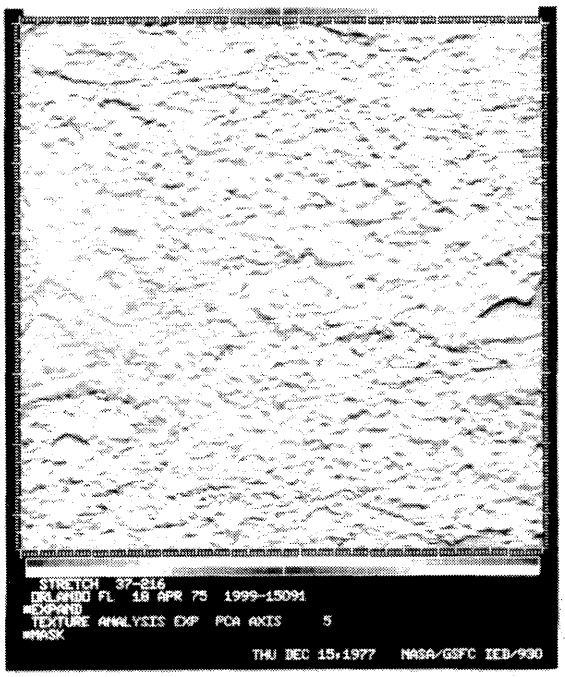


a

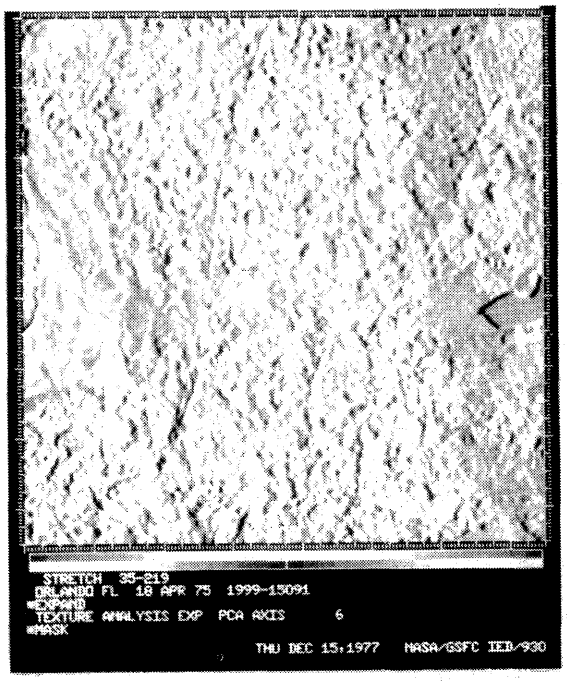
c

d

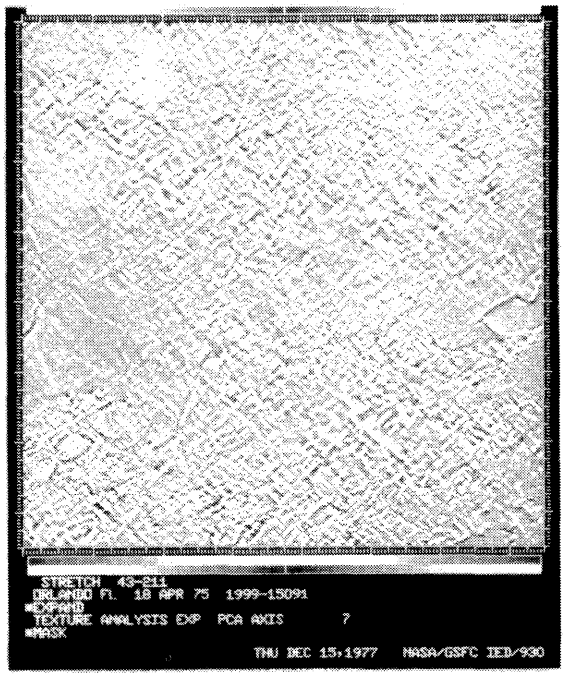
Figure 2—First eight principal components after textural feature augmentation



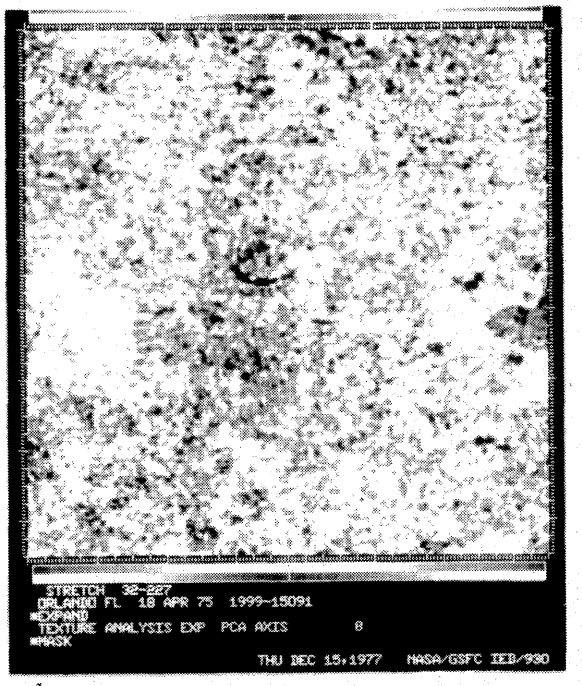
e



f



g



h

Figure 2 (continued)

EVEC 1	EVEC 2	EVEC 3	EVEC 4	EVEC 5	EVEC 6	EVEC 7	EVEC 8	EVEC 9	EVEC 10	EVEC 11	EVEC 12	EVEC 13	EVEC 14	EVEC 15	EVEC 16
-0.2138	-0.1900	0.1882	-0.1872	0.2228	0.2584	-0.1644	0.1665	-0.2677	0.0023	-0.0987	0.1105	-0.3558	-0.4234	0.3921	0.3479
-0.3174	-0.2397	0.2665	-0.2020	0.3302	0.3029	-0.1637	-0.0877	-0.2911	0.0311	-0.0714	0.1587	0.3199	0.2960	-0.3121	-0.3072
-0.2490	0.1682	0.2947	-0.2757	-0.0750	-0.1600	-0.3068	-0.3488	0.0939	-0.4325	0.3560	-0.3805	-0.0063	0.0119	-0.0038	0.1515
-0.2084	0.1552	0.2858	-0.2749	-0.2820	-0.2510	-0.1271	-0.2580	0.2380	0.4025	-0.2417	0.2141	0.0295	-0.0063	0.0124	-0.1074
-0.2210	-0.1856	0.1872	0.1742	0.1684	-0.1762	0.1548	0.2666	0.1651	0.1042	0.0809	-0.0733	-0.4502	0.4128	-0.3619	0.3532
-0.3218	-0.2253	0.2481	0.2720	0.2700	-0.2956	0.2723	0.0306	0.2700	0.1448	0.1065	-0.1918	0.2672	-0.2921	0.3042	-0.2844
-0.2431	-0.1642	0.2653	0.2689	-0.1252	0.0432	0.2859	-0.3885	0.0919	-0.3266	-0.4136	0.4360	0.0127	-0.0137	-0.0091	0.1737
-0.2014	0.3035	0.2453	0.2819	-0.3463	0.3529	0.2511	0.2075	-0.4044	0.1503	0.2859	-0.2539	-0.0073	0.0068	-0.0021	-0.0682
-0.2137	-0.1948	-0.1756	-0.2019	-0.2406	0.2620	0.1517	0.1483	0.2818	0.0153	-0.0413	-0.1401	0.3716	-0.3751	-0.4139	0.3473
-0.3197	-0.2382	-0.2343	-0.2267	-0.3256	0.3056	0.1379	-0.1016	0.2906	-0.0245	-0.1026	-0.1335	-0.3220	0.2870	0.3146	-0.3280
-0.2447	0.1676	-0.2696	-0.3071	0.1003	-0.1625	0.3280	-0.3485	-0.1221	0.3757	0.4164	0.3553	-0.0254	-0.0059	0.0256	0.1499
-0.2097	0.3596	-0.2611	-0.3010	0.2791	-0.2232	0.3277	0.3450	-0.2090	-0.3587	-0.3076	-0.1944	-0.0043	0.0101	-0.0285	-0.1142
-0.2193	-0.1894	-0.2167	0.1494	-0.2011	-0.1906	-0.1607	0.2260	-0.1916	-0.1168	0.0534	0.1083	0.4348	0.3960	0.3940	0.3543
-0.3219	-0.2246	-0.2673	0.2456	-0.2687	-0.3203	-0.2483	0.0258	-0.2750	-0.1505	0.0830	0.1681	-0.2536	-0.3101	-0.3112	-0.2937
-0.2403	0.1628	-0.2878	0.2686	0.1694	0.0274	-0.2741	-0.3685	-0.0857	0.3773	-0.3924	-0.4223	-0.0072	0.0193	-0.0233	0.1713
-0.1978	0.3624	-0.2760	0.2664	0.3268	0.3480	-0.2831	0.1955	0.3911	-0.1814	0.2942	0.2462	0.0022	-0.0217	0.0262	-0.0689

Figure 3—Eigenvectors defining the K-L transformation

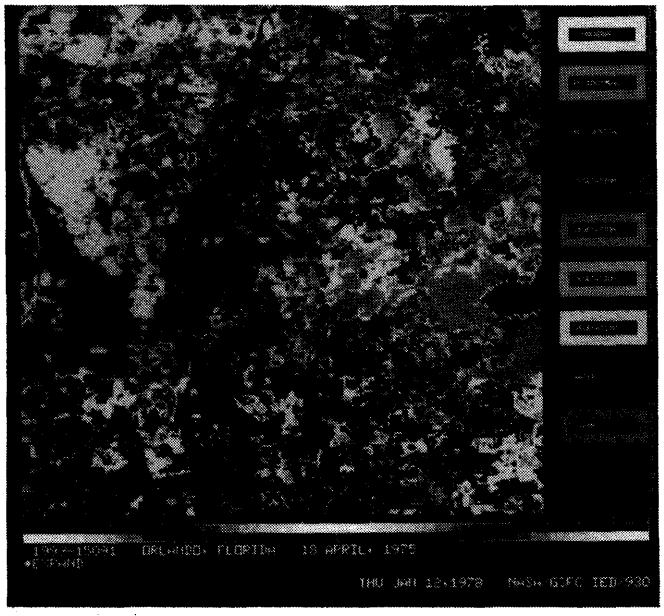


Figure 4—Classification map using raw Landsat channels

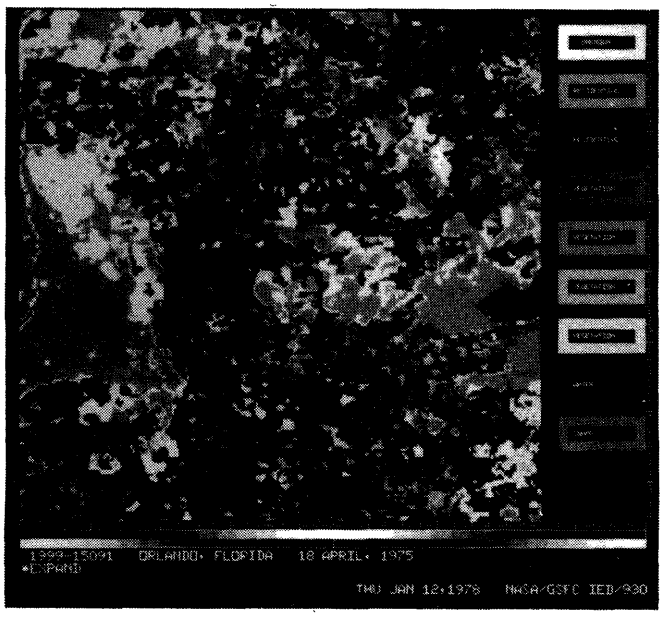


Figure 5—Classification map using the first six K-L transformed components after textural augmentation

features separately. The two resulting classification maps (CMs) are shown in Figures 4 and 5, respectively. The major differences between the two CMs are in the swamp (class 8)/vegetation (class 3) area on the left side of the image and in the increase of residential classifications in Figure 5 relative to Figure 4. The main increase in the residential classes comes from the locations in Figure 4 which were classified as vegetation. Although a ground-truth verification was not performed in this experiment, a slight overestimate, rather than underestimate, of the residential class is desirable (e.g., in census applications in which decisions on locations for manual data gathering are made based on the CMs). Also, the CM in Figure 5 is more uniform within each class than that in Figure 4. Figure 6 shows a confusion matrix between the two CMs, with the ith row, jth column element equal to the number of occurrences of the ith class in Figure 5 and the jth class in Figure 4. Defining the similarity measure between the two CMs to be the trace of the confusion matrix expressed as a percentage of the total number of pixels, it is found that the two CMs are 63.4 percent similar. Because the major differences between the CMs are in classes 3 and 8, the spectral and textural features were further investigated by generating plots of the spread of data values in each of the eight classes. These plots are shown for the spectral features in Figure 7 and for the first four textural features in Figure 8. It can be seen that, in the row data bands, the overlap between classes is greater than in the case of textural features, especially for classes (1, 2) and (3, 8). Because the separability of classes is better in the case of textural features, it is highly likely that the classifications in Figure 5 are more accurate than those in Figure 4.

CONCLUSIONS

Simple features using spectral and spatial information have been defined in this paper. A difference between the textural features defined herein and those defined in previous works

637	927	1	12	0	0	48	0
1769	17616	365	1654	1017	369	1866	0
12	215	2695	411	678	29	5	320
2	189	138	701	291	0	1	1
1	560	365	307	1959	156	0	29
0	337	120	3	583	5501	162	157
275	2888	102	129	286	2106	6838	14
0	0	3129	9	169	669	0	3677

Figure 6—Confusion matrix between classification maps

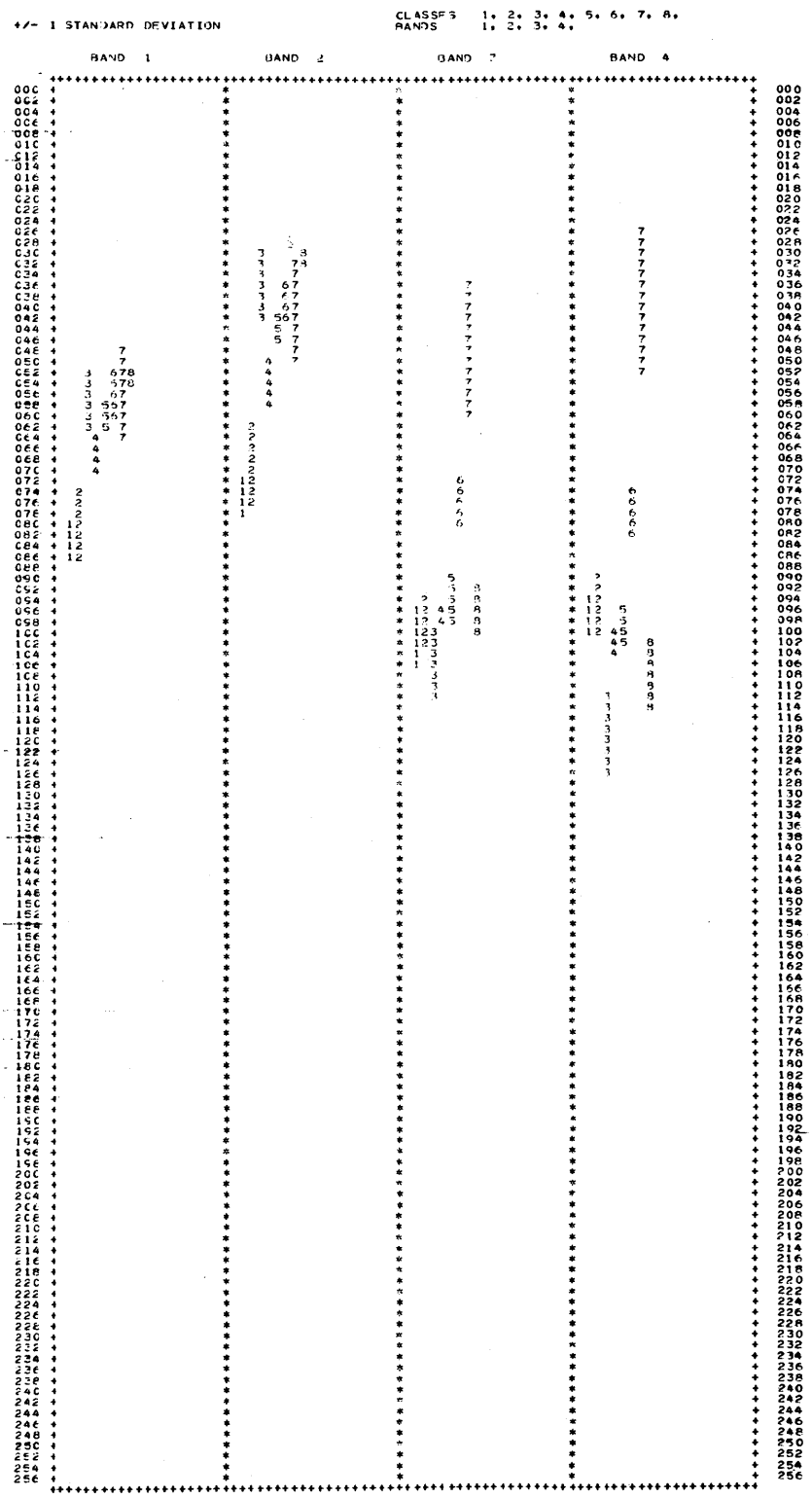


Figure 7—Distribution of channel values for spectral features

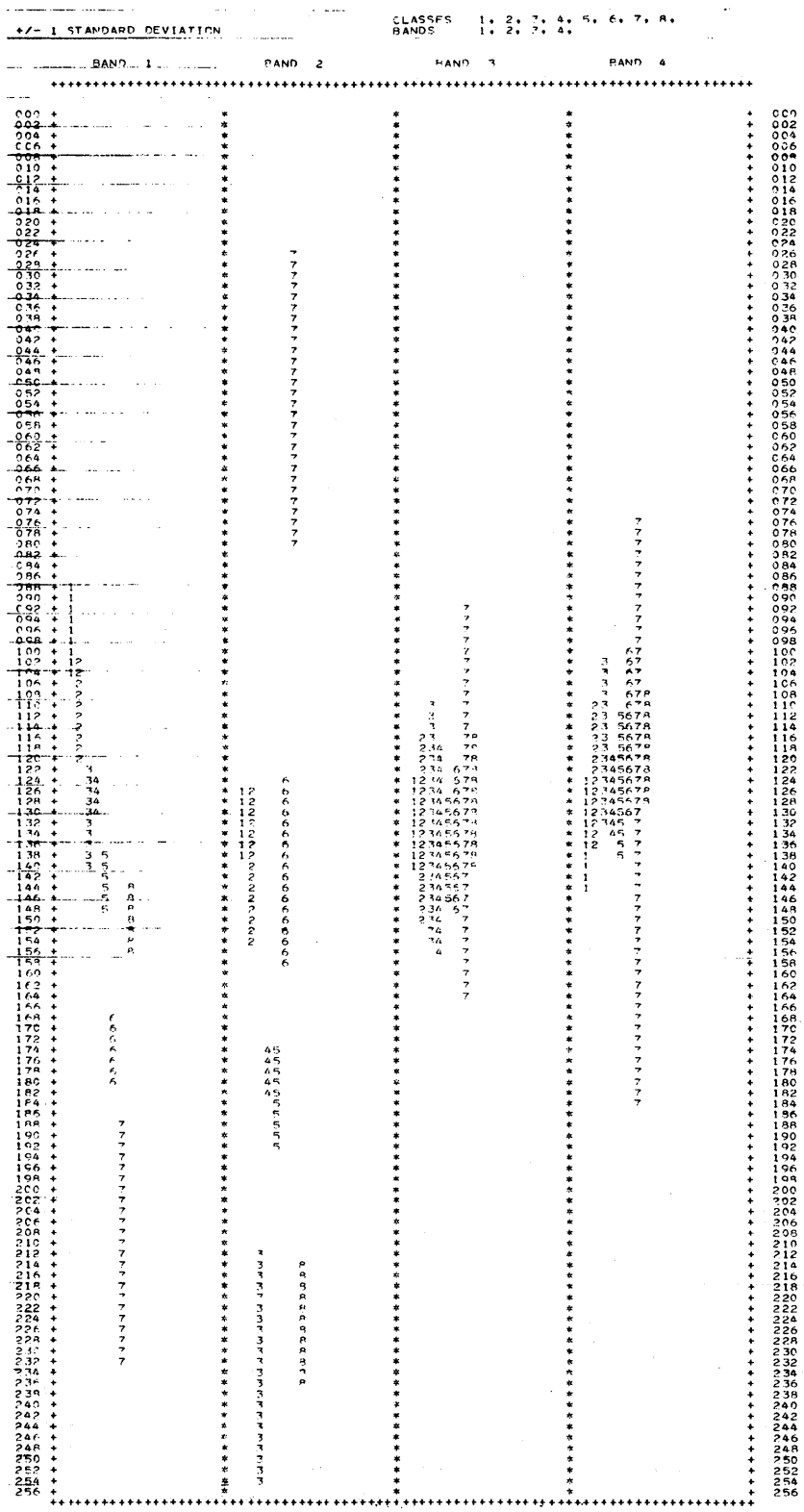


Figure 8—Distribution of channel values for K-L transformed components after textural augmentation

is that the features are defined here at every point in an image and the effective resolution is not reduced. An interpretation of the features extracted through linear combinations of the spectral reflectances of the nearest neighbors of a pixel is that the effects of the spatial variation in the immediate vicinity of the pixel are taken into account in making decisions on its classification. Experimental results are shown for a small subset of a Landsat scene using one of the feature definitions and the K-L transformation to reduce the dimensionality. The main differences between the classification map using spectral features and that using textural features are that the latter has a higher number of residential pixels and is smoother in appearance. An examination of the plot of the spreads of the spectral and textural feature values for the various classes reveals better separability of classes with textural features.

Further experiments with the features and dimensionality reducing transformations should include the use of symmetric feature vectors such as in Equations (3) and (4) and the use of a canonical transformation.

Combinations of three of the first few components obtained using the transformations can be used for producing color composites. Such color images will have an enhancement of the edges and are expected to be similar to the output of high-emphasis filters.

REFERENCES

1. *Proceedings of the International Symposia on Remote Sensing of Environment,* University of Michigan.
2. Jayroe, R. R. et al., "Classification Software Technique Assessment," National Aeronautics and Space Administration, NASA Technical Note D-8240, May 1976.
3. Weszka, J. S., et al., "A Comparative Study of Texture Measures for Terrain Classification," *IEEE Transactions on Systems, Man and Cybernetics,* Vol. SMC-6, No. 4, April 1976.
4. Ramapriyan, H. K., "Spatial Frequency Analysis of Multispectral Data," in *Remote Sensing of Earth Resources,* Vol. 1, edited by F. Shahrokhi (University of Tennessee Space Institute, Tullahoma, Tennessee), 1972.
5. Palgen, J. J. O., "Applicability of Pattern Recognition Techniques to the Analysis of Urban Quality from Satellites," *Pattern Recognition,* Vol. 2, 1970.
6. Haralick, R. M., et al., "Textural Features for Image Classification," *IEEE Transactions on Systems, Man and Cybernetics,* Vol. SMC-3, No. 6, November 1973.
7. Haralick, R. M., et al., "On Some Quickly Computable Features for Texture," *Proceedings of the Symposium on Computer Image Processing and Recognition,* Vol. 2, University of Missouri, Columbia, August 1972.
8. Haralick, R. M. and D. Anderson, "Texture-Tone Study With Applications to Digitized Imagery," University of Kansas Center for Research, Inc., CRES Technical Report 182-2, November 1971.
9. Haralick, R. M., et al., "Texture-Tone Study With Applications to Digitized Imagery," University of Kansas Center for Research, Inc., CRES Technical Report 182-4, December 1972.
10. Haralick, R. M. and K. Shanmugam, "Computer Classification of Reservoir Sandstones," *IEEE Transactions on Geosciences Electronics,* Vol. GE-11, October 1973.
11. Haralick, R. M. and R. Bosley, "Spectral and Textural Processing of ERTS Imagery," *Proceedings of the Third ERTS-1 Symposium,* Vol. 1, Washington, D.C., December 1973.
12. Haralick, R. M., "A Texture-Context Feature Extraction Algorithm for Remotely Sensed Imagery," *Proceedings of the 1971 IEEE Decision and Control Conference,* Gainesville, Florida, December 15-17, 1971.
13. Galloway, M. M., "Texture Classification Using Gray Level Run Lengths," *Computer Graphics and Image Processing,* Vol. 4, June 1975.
14. Young, T. Y. and T. W. Calvert, *Classification, Estimation and Pattern Recognition,* New York: American Elsevier Publishing Company, Inc., 1976.
15. Seal, H. L., *Multivariate Statistical Analysis for Biologists,* London: Methuen and Co. Ltd., 1964.

Applications of digital image processing techniques to problems of data registration and correlation

by WILLIAM B. GREEN
California Institute of Technology
Pasadena, California

INTRODUCTION

JPL's Image Processing Laboratory (IPL) has been processing imagery using digital computers for approximately ten years. During that period, the computer capability at the IPL has developed from a single job batch oriented system with no volatile image displays to a multi-task environment accommodating both batch processing and several interactive tasks simultaneously, supporting a variety of general purpose and special purpose interactive image display systems. The sophistication of the various applications has also developed from simple subjective enhancement of planetary imagery to complex applications requiring correlation of remotely sensed imagery with auxiliary data and with non-imaging data bases. This paper describes the evolution of the IPL facility, and illustrates the increasing complexity of the image processing tasks with examples of various applications from the planetary program and the earth resources activities that have utilized the IPL facility.

HISTORICAL BACKGROUND

Ten years ago at JPL, digital image processing of lunar images from Ranger and Surveyor was performed using a central general purpose computer in a batch environment. An analyst submitted his job and often had to wait several days to view the result, since the job had to be run in competition with other tasks using the same computer and the output image had to be recorded on film, processed and printed before he could see the results of processing.

In 1968, IPL acquired an IBM 360/44 computer system that was dedicated to image processing applications. This computer, and the 44 PS operating system, were capable of supporting a single user at any given time; the analyst again had to wait to view the result of his processing until film recording, film processing and printing had occurred. At that time, there were no commercially available image display systems that could be easily interfaced to a digital computer so the IPL developed a core-refreshed display (CRD) system. The CRD used the 360/44 core memory as its refresh memory, and the display was capable of displaying a 256 picture element square image with 4 bits of intensity quantization. The core memory of the 360/44 was read every 1/30 of a second to refresh the video display at video rates. The CRD made it possible to display imagery as soon as it had been processed, although the display was limited to 256 square resolution and 4 bits of intensity information. It had the obvious disadvantage of precluding any processing while image display was occurring, since the 360/44 memory was totally committed to image display refresh buffering whenever an image was being viewed on the CRD. Nevertheless, it represented the first interactive image processing capability at the IPL, and substantially shortened the turnaround time for subjective enhancement processing.

The 360/44 operating system and hardware display support improved between 1968 and 1975. A new image display system employing lithicon storage tube refreshed display systems was implemented to support the Mariner 9 Mars orbital mission in 1972. By 1974, it was possible to run two or three jobs on the system in a timesharing mode, using foreground and background partitions. However, processing became severely limited by the constraints of the 360/44 system as the size of the imagery returned by the planetary missions increased, as the number of images returned from each mission increased dramatically, and as large multispectral images became available from the LANDSAT satellites. In 1975, a decision was made to upgrade the entire IPL computer configuration and associated display systems to meet the following objectives: (a) increase batch processing capacity by at least a factor of three, (b) enable simultaneous interactive image processing to occur at the same time that batch processing was occurring, (c) enable at least two interactive image processing users to be active at any given time.

The next section describes the current IPL computer and display configuration, and the remaining sections discuss a variety of applications for which the system has been used.

CURRENT IPL CONFIGURATION

The current central computer in the IPL is an IBM 360/65 with a megabyte of memory, eight tape drives, and 900 mbytes of on-line disk storage; 800 of the 900 mbytes of on-line storage consists of Memorex 3670 high speed disk drives.

The 360/65 operating system is OS/MVT, and TSO is used to support timesharing operations. The Informatics MARK IV data base management system is also available, and is used to catalog processed imagery produced by the IPL each day. In addition, MARK IV is used to maintain catalogs of auxiliary data relating to each image processed by the IPL. For example, an image of Mars recorded by the Viking Lander will have a variety of parameters associated with it, such as local Mars time at which the exposure was made, camera gain and offset settings, solar illumination angle at the time of exposure, etc. MARK IV is used to construct, update and search these catalogs.

The interactive processing is supported by a variety of terminals and display systems. A Digital Equipment Corporation PDP 11/40 computer is interfaced to a channel on the 360 via a DX-11B channel interface. The PDP 11/40 supports several dial-up and hard-wired interactive terminals, and three image display systems. The interactive terminals include both IMLAC graphics/CRT terminals and TI Silent 700 terminals. The current IPL configuration is shown in Figure 1.

The image display systems are all commercially available equipment. They include a RAMTEK display system that accommodates 6 bit black and white imagery up to 640×512 elements. The RAMTEK also has two one-bit graphics overlay planes and two trackball/cursor units. Each of the RAMTEK output signals (video only, video plus either graphics plane, or either of the graphics planes) are available as video signals that may be separately accessed and displayed. Thus more than one user can use the RAMTEK system at a given time. For example, one user can access the black and white video image with one graphics plane overlaid, while a second user can access just the second graphics plane. All video signals are routed into a video network that includes several black and white monitors distributed in two user area locations at IPL, and a user selects the video signal he wishes to utilize and displays that signal on a monitor near his terminal. One view of one of the interactive user stations is shown in Figure 2, and a user interacting with an image display is shown in Figure 3.

The other image display systems consist of two COMTAL display systems. A COMTAL 8003 system provides 512×512 resolution for six bit images, and includes graphics overlay planes and trackball/cursor unit. The COMTAL

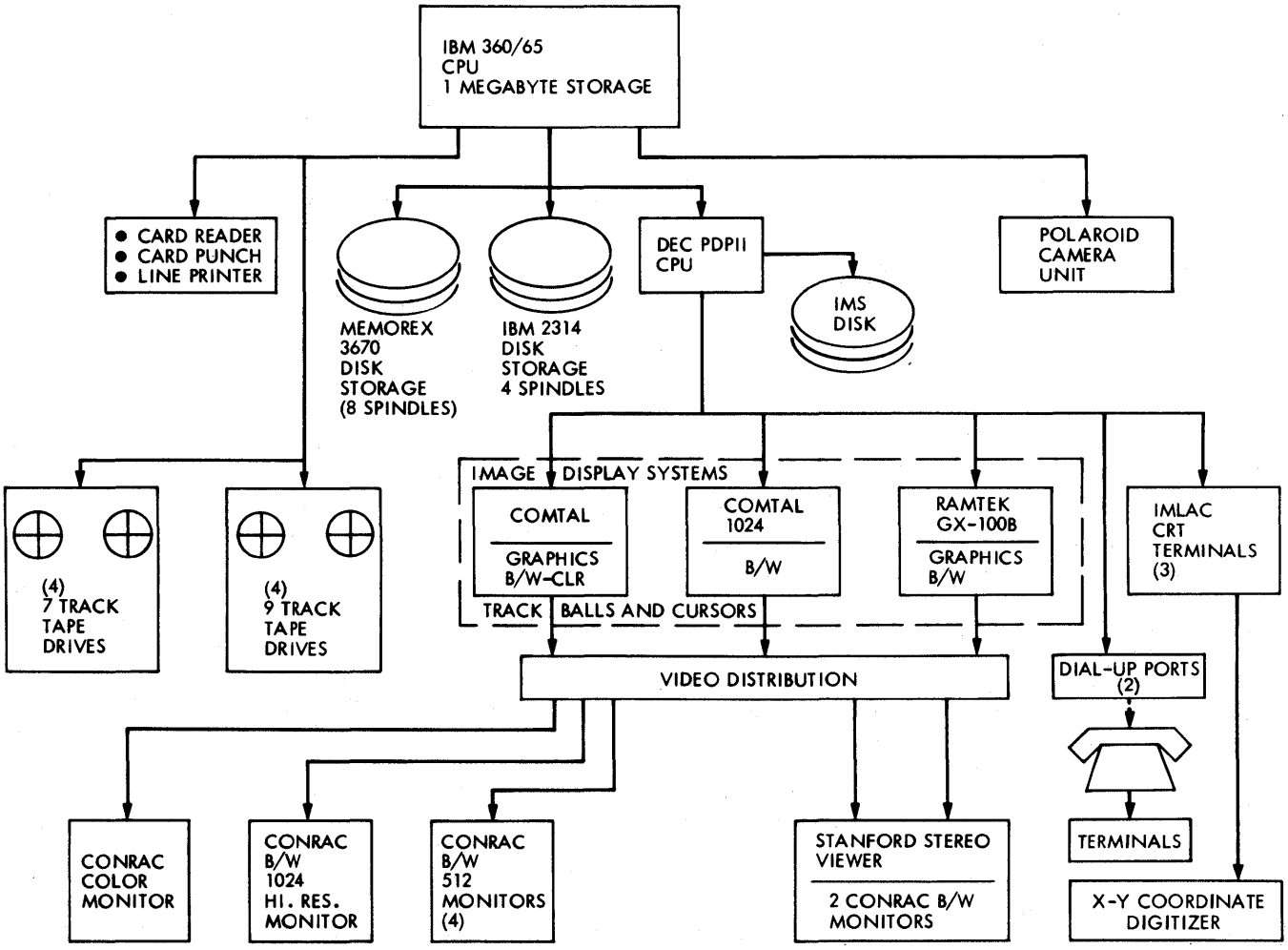


Figure 1—JPL Image Processing Laboratory computer configuration

Figure 2—User area station at JPL's Image Processing Laboratory. The user seated at the IMLAC terminal can select video signals to be routed to the display monitors using the video switching box on the right side. Audio communication with the computer operator is also provided. (JPL Photo Lab Negative No. 324-2009Ac)

8003 provides either a single color image at 512×512 resolution or three black and white images at the same resolution. Again, each output video signal is separately accessible, and can be routed to any monitor in the user areas.

A COMTAL 1024 system provides capability for display of black and white imagery at 1024×1024 resolution. Because of the high bandwidth, the video output of this device is routed to a single specially modified monitor. The 1024 system also includes graphics overlay and trackball/cursor unit.

A complete description of the IPL computer configuration and the interactive capabilities supported by the PDP 11/40 computer is contained in Reference 1.

Figure 3—Interactive extraction of lakes from LANDSAT imagery achieved through utilization of the COMTAL display and associated trackball/cursor unit. (JPL Photo Lab Negative No. 324-2287Ac)

GEOMETRIC TRANSFORMATION

IPL has been performing digital geometric transformation of imagery for approximately ten years. The earliest applications included removal of camera system induced geometric distortion[2–4] and projection of planetary imagery to standard cartographic projection.[5] More recently, geometric transformation has become an important tool in registration of imagery and non-imaging data represented as imagery to standard georeferenced coordinate systems, and in registration of multiple images of the same surface area recorded at different times.

Geometric transformation of digital imagery begins by locating a set of tiepoints within the original image and defining the locations of those tiepoints in the output image. The transformation of the tiepoints may be determined by a mathematical algorithm or by analysis of the imagery itself. For example, image transformation to standard mapping projections can be defined by establishing a uniform grid of tiepoints in the output image aligned with a longitude/latitude coordinate system, and then mathematically computing the locations of those tiepoints in the untransformed image based on a model of the spacecraft location and imaging sensor geometric characteristics. The locations in the input image corresponding to points between tiepoints in the output image is computed by an interpolation procedure.

The intensity of picture elements in the output image that do not fall on exact points in the input image is computed by performing an intensity interpolation (or resampling) in the input image. The process is shown in Figure 4. Normally, the intensity of each sample on each line of the output image is computed in turn, starting at the upper left corner of the image and working down to the lower right corner. For each output sample, the corresponding location in the input image is computed, and the intensity of that sample is computed based on an interpolation scheme if the location in the input image is not exactly at a discrete image sample point. Typical interpolation schemes include nearest neighbor, bilinear interpolation, and cubic spline.

An example of geometric transformation is shown in Figure 5. A Viking Orbiter image of Mars is shown before and after orthographic projection. The unprojected image shows the large crater to have an elliptical shape, due to distortion

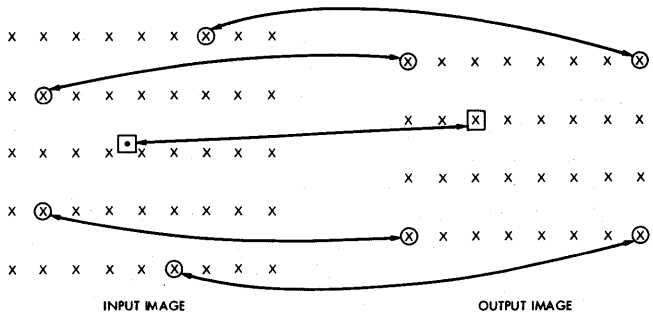


Figure 4—Digital geometric transformation

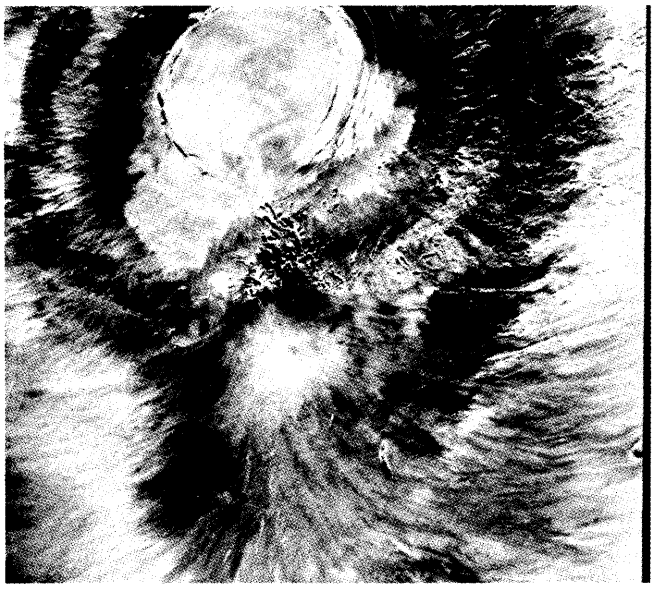


Figure 5—Left—Contrast enhanced version of a Viking Orbiter image of a large Martian crater. The crater is ellipsoidal, due to spacecraft viewing geometry.

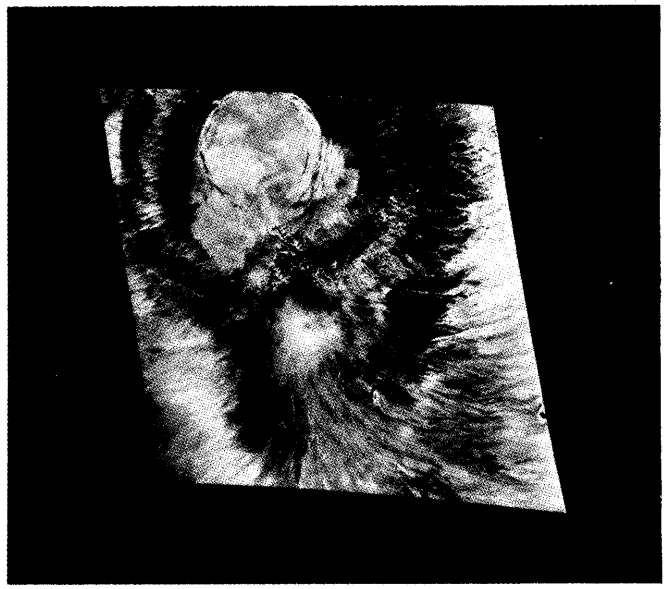


Figure 5—Right—The same image after orthographic projection. Shape distortion due to viewing geometry has been removed. (IPL Pic ID 76/09/09/173047 and 76/09/09/175425)

introduced because the image was recorded when the camera was at an oblique angle relative to the local surface normal. The orthographic projection is performed to depict the images as if the camera were viewing the scene from directly above the center of the field of view, and the transformation is defined based on spacecraft position and camera orientation data returned in the spacecraft telemetry.

AUTOMATED IMAGE REGISTRATION

It has frequently been possible to perform image registration in an automated manner in the NASA planetary program. Typically, each planetary spacecraft records a large number of images of a planetary surface, and the science investigators require construction of photomosaics showing large portions of the surface of a planet in a single photographic product. In most cases, it is possible to perform standard mapping projections by computing the geometric transformation for each individual image based on auxiliary data returned with each image that describes the spacecraft position relative to the planet and the camera viewing angle. The transformation is computed based on the mathematical model of the planet (generally an oblate spheroid is used as the planet model), the camera system optical components, and the spacecraft and camera viewing geometry.

IPL produced over 1200 images projected to an orthographic projection during the Mariner 9 Mars orbital mission. Each image was individually corrected for camera system geometric distortion, enhanced, and then projected to an orthographic projection. The projected images were then photographically scaled and used to construct the first global photomosaic of a planet ever constructed from remotely sensed imagery. The photomosaic was constructed on a three-foot diameter globe, and a portion of the photomosaic is shown in Figure 6.

The Mariner 10 flyby mission to Venus and Mercury provided several thousand images of the surface of Mercury. IPL produced photomosaics of many standard mapping quadrants on the planet's surface. Figure 7 shows one of the mapping quads produced from over fifty Mercury images from Mariner 10. Again each image was individually processed to remove camera system induced distortion, enhanced, and projected to Mercator projection. Other map-

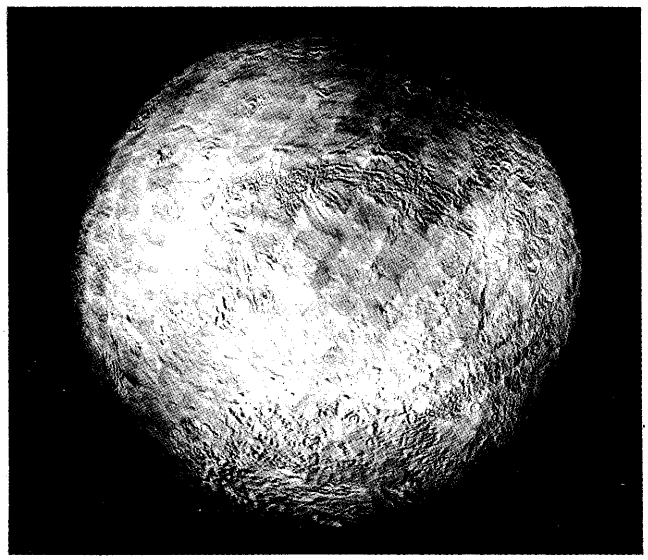


Figure 6—One hemisphere of the photomosaic globe of Mars produced using over 1200 enhanced and orthographically projected images from the Mariner 9 Mars orbital mission. (JPL Photo Lab Negative No. 320-284B)

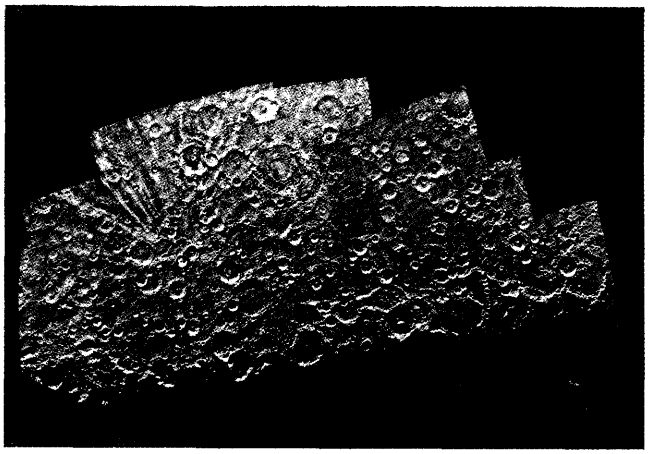


Figure 7—Computer generated mosaic of the region near the south pole of Mercury, constructed from over twenty separate Mariner 10 vidicon images. A special scene dependent high pass filter has been applied to bring out local detail throughout the scene, and especially near the terminator. (JPL Photo Lab Negative No. 324-2306)

ping projections in standard use at IPL include Lambert Conformal Conic and Polar Stereographic.

In both of the cases described above, the navigation data was of sufficient quality to enable mosaicking of automatically projected imagery. After each individual image was projected, it was not necessary to refine the projection transformation to achieve a better match of the imagery. However, there are instances where precise registration to less than a picture element across the entire area of interest is required. In those cases, it is often necessary to perform manual or interactive processing to refine the transformations that are performed. Typical applications of this type are discussed in the next section.

SEMI-AUTOMATED IMAGE REGISTRATION

IPL has developed the capability to perform interactive refinement of image registration for specific applications. An example of an application requiring registration accuracies of less than one picture element is image differencing. For example, the Mariner 9 Mars Orbiter provided repetitive coverage of portions of the Martian surface during 1972. Determination of surface feature variation was one scientific objective of that mission. It was possible to achieve gross (accuracy $\geq$1 picture element) registration of two images of the same portion of the surface taken at different times during the mission by performing standard mapping transformations. However, change detection can be confusing if there is misregistration on the order of one picture element, since changes are masked by overall misregistration when a difference image is produced. It was necessary to refine the registration using an interactive display and supporting software. The software originally developed for Mariner 9 has been extended and refined for later applications, but the general principles remain the same.

An analyst selects a feature in one of the two images he wishes to register. He then identifies the same feature in the second image. The program uses his second location in the second image as the center of a search area, and performs correlation calculations within a local moving window in the second image. A point of maximum correlation is selected as the location in the second image that corresponds to the selected location in the first image, and the offset of that point relative to the location in the first image is retained. The analyst proceeds to select a set of features and performs the same routine with each point, so that a set of offset vectors is ultimately available throughout most of the region of overlap. When an adequate number of feature offsets has been acquired, a geometric transformation is defined that will map the second image onto the first image. Normally, if the two images have been processed to a standard mapping projection prior to the identification of feature points, the second geometric transformation is a small refinement. The result is a registered pair of images where registration has been achieved to less than one picture element accuracy.

Figure 8 shows one of the output products from the Mariner 9 image differencing task at IPL. The top two images are segments of Mariner 9 imagery of the same part of the surface recorded approximately two weeks apart. The lower left image is an unenhanced difference picture, and the lower right image is a contrast stretched version of the difference picture. Any differences in intensity between the two images is displayed as either whiter or blacker than mid-gray in the difference image.

If the two images had been misregistered by more than a picture element, all features in the scene would appear in the difference image as both black and white detail. For example, the crater edge would appear as a double image, one black and the other white. Because the registration is precise in the example shown in Figure 8, the crater edge appears only once in the difference image, and the difference image correctly portrays the change in crater edge definition and albedo that occurred as the great Martian dust storm of 1972 continued to clear during the two-week period in which these images were recorded.

REGISTRATION OF IMAGING AND NON-IMAGING DATA

The registration of non-imaging data to imaging data has always been important in digital image processing applications at IPL. Processing of planetary imagery often required knowledge of auxiliary data relating to each recorded image. For example, generation of geometric transformation parameters to transform an image into a standard mapping projection requires a knowledge of the spacecraft position and the camera viewing angle at the time the image is recorded. It is also often desirable to remove the shading effects in planetary imagery that are caused by variation in solar illumination and viewing angles from image to image, and removal of these effects is performed using auxiliary ephemeris information and spacecraft position and view angle data.

The planetary program initially provided the motivation for geographically referenced image data bases. It was nec-

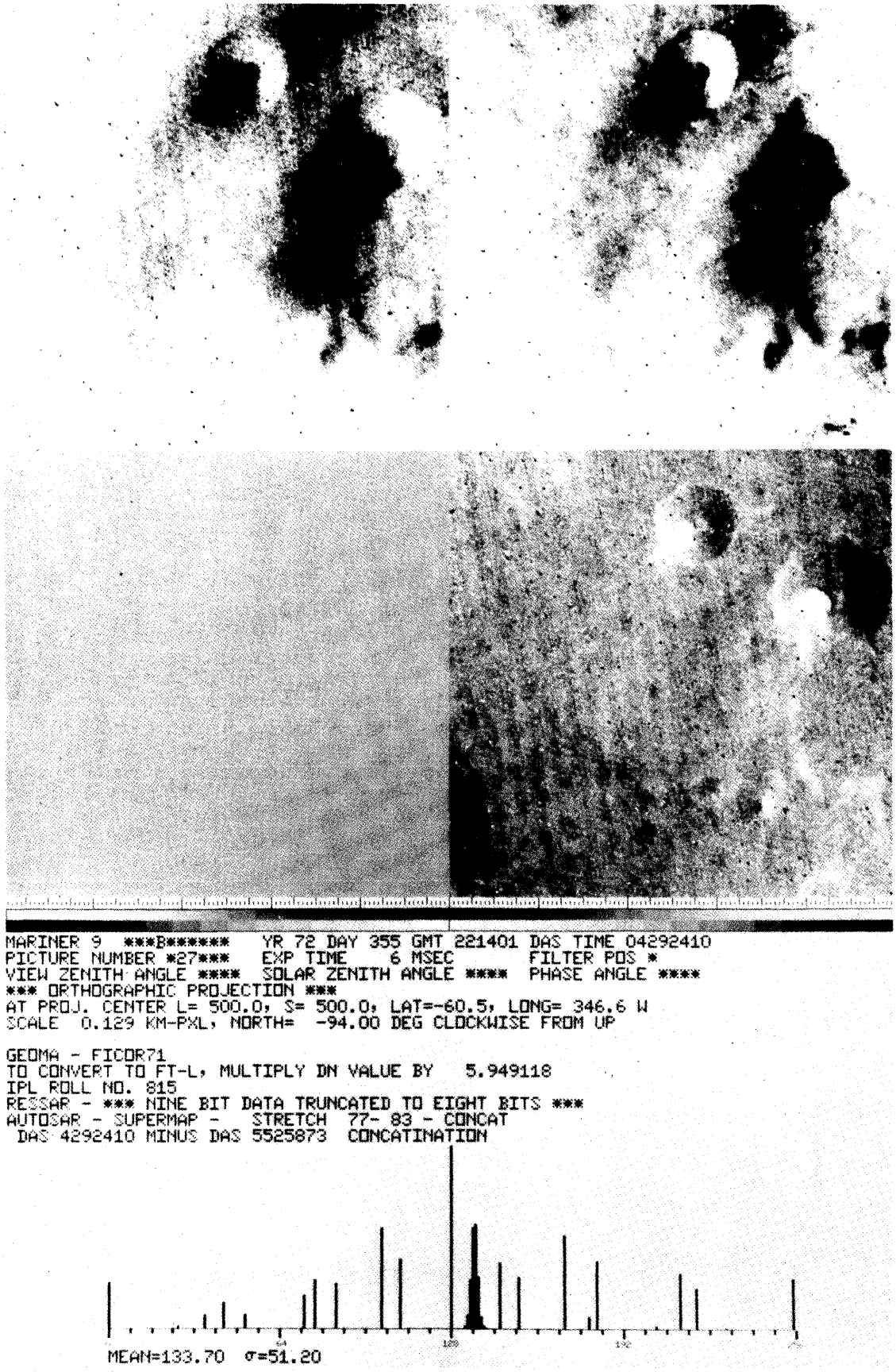


Figure 8—An example of image differencing. Top two images are segments of Mariner 9 images of Mars recorded approximately two weeks apart during clearing of the great dust storm of 1972. The two segments are in precise registration. The bottom left image is an unenhanced difference image, and the bottom right image is a contrast enhanced difference image. Changes in albedo in the center of the crater show as white areas in the difference image. (JPL Photo Lab Negative No. 324-2293A)

essary to index each image based on the longitude and latitude coordinates of the surface area recorded in each individual image. Computerized data bases were created for imagery of Mars, Venus and Mercury that enabled interactive interrogation to determine surface coverage as a function of a variety of parameters. It is possible at IPL to perform a search to locate all images taken of a particular portion of the planet surface that were taken under a particular set of conditions. Thus an analyst can interactively query a picture catalog to answer the following type of question:

"What Viking Orbiter images exist that show the area within 10 degrees of latitude 65° west longitude 120° after 3:00 p.m. local Martian time that were taken through the red spectral filter? Limit the search to those images recorded after the 35th day of the Orbital mission."

The LANDSAT satellites have increased the emphasis on correlation of geographically encoded data bases. There are literally hundreds of existing geographically encoded data bases in routine use throughout the country by a variety of federal, state and local government agencies. One of the most familiar examples is the data base maintained by the United States Census Bureau. LANDSAT satellites provide repetitive multispectral imagery of the earth's surface, and have been used to monitor agricultural parameters, geological properties, and land use patterns throughout the country, to list just a few applications. In almost every case, there is a need to relate results obtained from analysis of LANDSAT imagery to existing geographically referenced data bases. In recognition of this need, IPL has developed the Image Based Information System (IBIS).[6,7] This system utilizes the geometric transformation capabilities originally developed for the planetary program to correlate imaging and non-imaging data bases. Figure 9 shows the results of a standard thematic classification of land use in the Portland, Oregon area.[9] Each picture element in a LANDSAT multi-

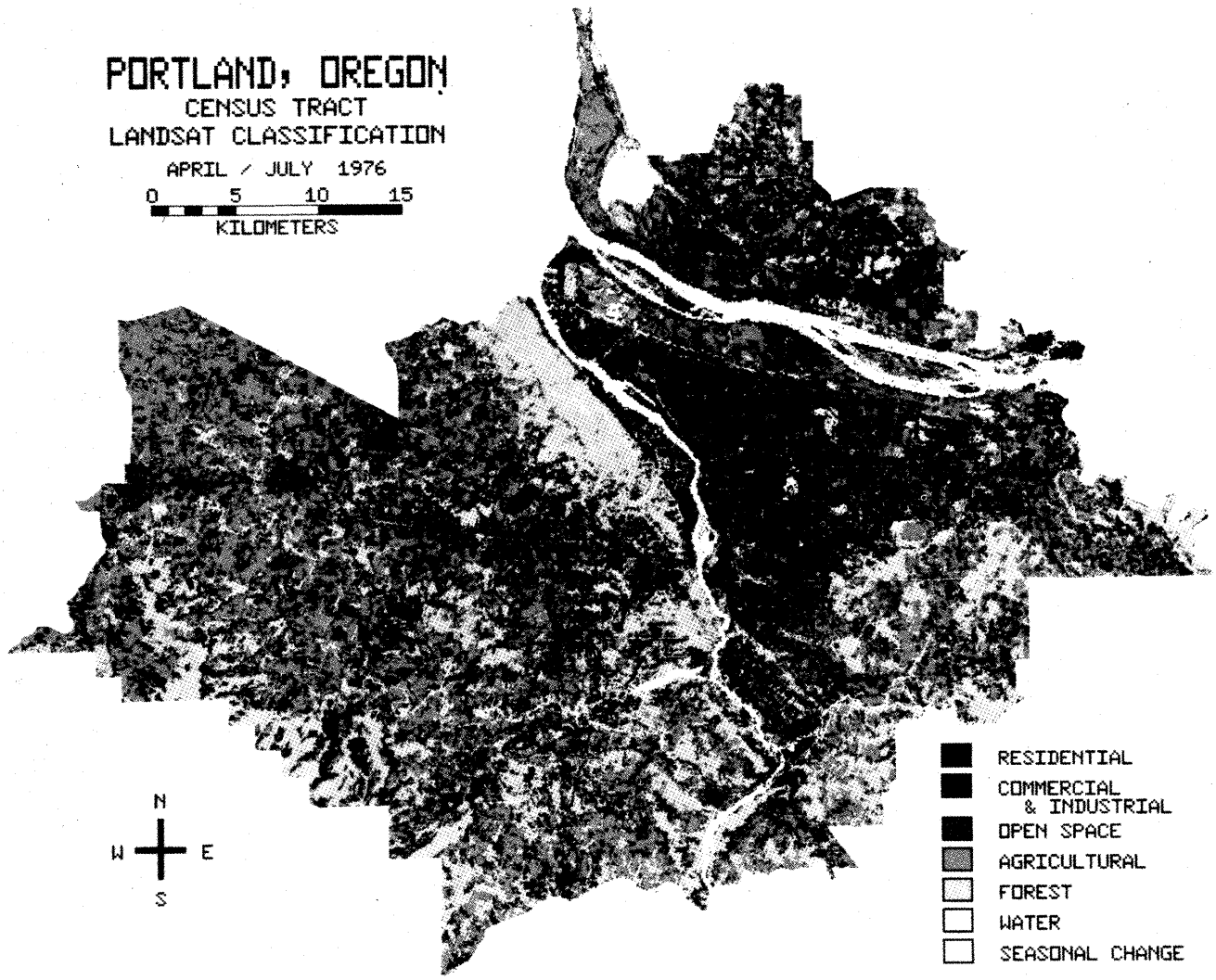


Figure 9—Thematic map of Portland, Oregon showing land use patterns determined from multispectral classification of LANDSAT imagery. The white overlay indicates U. S. Census tract boundaries registered with the LANDSAT image.

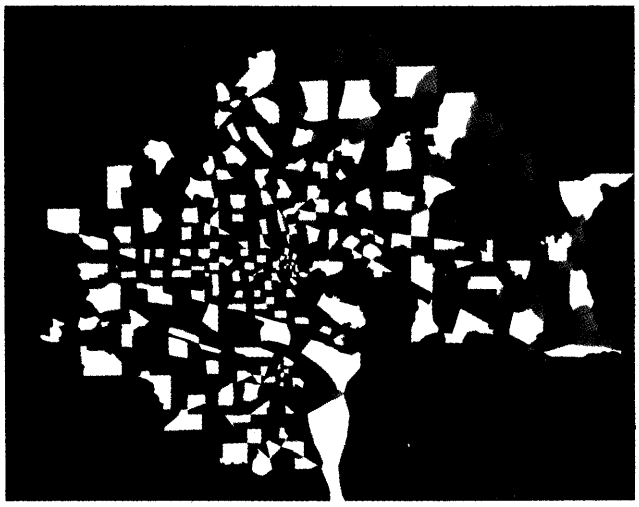


Figure 10—District image created from tabular data base containing traffic zone boundaries in Portland, Oregon. The district image display product is created from the digital district image using a five-color algorithm;[9] the district image is registered to the LANDSAT thematic map shown in Figure 9.

spectral image is assigned to a particular class of land use based on the relative spectral response in each of the four spectral bands recorded by LANDSAT. Training areas for which ground truth is known are often used to define the spectral signature corresponding to each desired land use category.

The graphical overlay visible in Figure 9 represents the boundaries of the Census tracts for the Portland area. The Census information has been registered to the LANDSAT image using the same registration techniques previously described for the planetary applications. The U. S. Census Bureau provides geographically indexed graphical files on magnetic tape. The data base provided by the Census Bureau includes the geographic location of each of the vertices visible in Figure 9, and a set of parameters for each census tract (e.g., population, age distribution). The Census tabular file is read, and a district image is created. The district image is registered to the LANDSAT image, and contains picture elements that are coded to indicate the census tract identification number in which that picture element lies. An image display product of the district image is often created as a visual check on the district image. Figure 10 shows a district image for the Portland area, based on traffic zone definitions within the city of Portland rather than the Census data. It is important to note that often several overlapping geocoded data bases exist for a single area. In the case of the Portland application, we see that both Census data and traffic zone data bases exist, and that the boundaries of the traffic zones do not correspond to Census tract boundaries. This is one reason that an image-based correlation procedure is useful. It is possible to perform cross-correlations between multiple overlapping geographically referenced data bases and remotely sensed multispectral imagery using the IBIS system; the tabulation is achieved by simple line by line summation using the multiple registered imagery. As an example, Figure 11 contains a portion of a tabulation of land use by traffic zone in Portland. The land use data is obtained from the thematic map shown in Figure 9, and the tabulation by traffic

PAGE 1.014

PORTLAND, OREGON
LAND USE ACREAGES BY TRAFFIC ZONES

		RESIDENTIAL		COMM/IND		OPEN SPACE		AGRICULTURAL		FOREST		WATER	
ZONES	AREA	AREA	PCT	AREA	PCT	AREA	PCT	AREA	PCT	AREA	PCT	AREA	PCT
704.00	885	175	19.8	0	0.0	12	1.4	443	50.1	253	28.6	0	0.0
705.00	1939	472	24.4	66	3.4	12	0.6	731	37.7	652	33.7	5	0.2
706.00	1886	122	6.5	0	0.0	34	1.8	1041	55.2	689	36.5	0	0.0
707.00	681	138	20.2	3	0.5	5	0.7	369	54.1	165	24.2	2	0.3
708.00	1065	470	44.2	46	4.3	2	0.2	359	33.7	188	17.6	0	0.0
709.00	749	311	41.5	48	6.4	2	0.3	231	30.8	157	21.0	0	0.0
710.00	1709	392	23.0	55	3.2	15	0.9	688	40.2	559	32.7	0	0.0
711.00	1871	348	18.6	15	0.8	33	1.8	879	46.9	595	31.8	1	0.1
712.00	2224	243	10.9	3	0.2	117	5.3	1067	48.0	793	35.7	1	0.1
713.00	1280	72	5.7	1	0.1	38	3.0	647	50.5	521	40.7	0	0.0
714.00	4819	149	3.1	6	0.1	60	1.2	1538	31.9	3057	63.4	1	0.0
715.00	2370	139	5.9	1	0.0	28	1.2	925	39.0	1277	53.9	0	0.0
716.00	3381	51	1.5	0	0.0	39	1.2	1099	32.5	2157	63.8	28	0.8
717.00	4144	30	0.7	0	0.0	27	0.7	720	17.4	3361	81.1	2	0.1
718.00	2895	63	2.2	0	0.0	18	0.6	540	18.7	2164	74.8	98	3.4
719.00	1244	19	1.5	0	0.0	12	1.0	288	23.2	832	66.9	53	4.2
720.00	1282	38	3.0	6	0.4	19	1.4	341	26.6	811	63.3	30	2.3
721.00	296	39	13.2	3	1.0	13	4.2	184	62.2	56	19.0	1	0.4
722.00	1501	190	12.7	25	1.7	0	0.0	173	11.5	385	25.7	721	48.0
723.00	112	14	12.8	85	76.5	0	0.0	5	4.5	0	0.0	7	6.3
724.00	119	60	50.4	11	9.4	0	0.0	23	19.6	16	13.1	9	7.5
725.00	180	73	40.2	7	3.9	0	0.0	91	50.6	10	5.3	0	0.0
726.00	212	62	29.1	36	16.9	0	0.0	70	33.1	44	20.8	0	0.0
727.00	809	206	25.5	70	8.7	0	0.0	214	26.4	87	10.8	231	28.6
728.00	368	62	16.9	13	3.5	0	0.0	64	17.5	17	4.5	212	57.6
729.00	246	73	29.9	2	0.8	0	0.0	68	27.8	102	41.6	0	0.0
730.00	1857	62	3.3	2	0.1	19	1.0	548	29.5	1218	65.6	1	0.1
731.00	735	148	20.1	72	9.8	0	0.0	321	43.7	69	9.4	125	17.0
732.00	775	52	6.7	2	0.3	2	0.3	311	40.1	400	51.6	0	0.0
733.00	382	23	6.0	1	0.3	37	9.8	203	53.1	118	30.9	0	0.0
734.00	56	37	66.6	9	16.5	0	0.0	10	16.9	0	0.0	0	0.0
735.00	78	68	87.0	2	2.9	0	0.0	8	10.1	0	0.0	0	0.0
736.00	53	32	59.7	20	38.4	0	0.0	1	1.9	0	0.0	0	0.0
737.00	66	37	55.9	10	15.4	0	0.0	12	18.6	7	10.1	0	0.0
738.00	76	72	95.0	1	1.5	0	0.0	2	2.3	1	1.2	0	0.0
900.00	123	50	41.1	72	58.9	0	0.0	0	0.0	0	0.0	0	0.0
901.00	94	77	82.3	6	6.1	0	0.0	9	9.6	2	2.0	0	0.0
902.00	583	307	52.6	54	9.3	4	0.6	180	30.9	38	6.6	0	0.0
903.00	763	220	28.9	144	18.9	0	0.0	274	35.9	3	0.4	121	15.9
904.00	5620	383	6.8	596	10.6	55	1.0	331	5.9	3414	60.8	685	12.2
905.00	4670	1786	38.2	243	5.2	195	4.2	1631	34.9	804	17.2	9	0.2
906.00	85	12	13.5	74	86.5	0	0.0	0	0.0	0	0.0	0	0.0

Figure 11—Tabulation of land use by traffic zone in Portland, created by using a registered district image in conjunction with the thematic map.

zone is achieved by correlation of the traffic zone district image with the thematic map.

The IBIS system has recently taken advantage of the interactive capabilities of the current IPL computer system. In particular, the image display systems provide a convenient way to perform precise registration of non-imaging and imaging data bases. As an example, the Census Bureau tabular data base often contains errors, resulting in failure of tract boundary lines to meet at vertices or generation of erroneous tract boundary lines and an erroneous district image when the tabular data is converted to image format. An interactive display system with graphics overlay capability provides an efficient tool for editing of the census data base to correct or remove occasional erroneous data entries.

SUMMARY

This paper has provided an overview of the evolution of the computer configuration at JPL's Image Processing Laboratory, and a summary of the development of techniques for geometric transformation of digital imagery. Image registration techniques originally developed for the planetary program have been described, and have become an important component of the registration of non-imaging geographically encoded data bases with multispectral imagery returned by earth observations satellites. Registration of existing geocoded data bases with LANDSAT imagery will continue to be important if the LANDSAT data is to be truly useful to the user community.

ACKNOWLEDGMENTS

The recent upgrade of the IPL computer facility was performed by the IPL Operations Group under the direction of Joel Seidman. The interactive display support software was developed under the direction of Paul Jepsen. The image registration software at IPL has evolved through the efforts of Tom Rindfleisch, Arnold Schwartz and Jean Lorre. Dr. Nevin Bryant and Dr. Albert Zobrist have been responsible for leading the development of the IBIS system; they are members of the Earth Resources Applications Group supervised by Richard Blackwell. The Mariner 9 image differencing example was provided by Paul Jepsen, and the Mariner 10 computer-generated mosaic was produced by Joel Mosher of our Space Image Processing Group supervised by Donald Lynn.

This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology under contract NAS 7-100, sponsored by the National Aeronautics and Space Administration.

REFERENCES

1. Jepsen, P. L., "The Software/Hardware Interface for Interactive Image Processing at the Image Processing Laboratory of the Jet Propulsion Laboratory," *Proceedings of the Digital Equipment Computer Users Society*, December 1976.
2. Rindfleisch, T. C., J. A. Dunne, H. J. Frieden, W. D. Stromberg and R. M. Ruiz, "Digital Processing of the Mariner 6 and 7 Pictures," *J. Geophys. Res.* 76 (2), 1971, pp. 394-417.
3. Green, W. B., P. L. Jepsen, J. E. Kreznar, R. M. Ruiz, A. A. Schwartz and J. B. Seidman, "Removal of Instrument Signature from Mariner 9 Television Images of Mars," *Appl. Opt.* 14, 1975.
4. Soha, J. M., D. J. Lynn, J. J. Lorre, J. A. Mosher, N. N. Thayer, D. A. Elliott, W. D. Benton, and R. E. Dewar, "IPL Processing of the Mariner 10 Images of Mercury," *J. Geophys. Res.* 80 (17), June 1975.
5. Gillespie, A. R. and J. M. Soha, "An Orthographic Photomap of the South Pole of Mars from Mariner 7," *Icarus* 16, 1972, pp. 522-527.
6. Zobrist, A. L., "Elements of an Image Based Information System," *Proceedings of the Caltech/JPL Conference on Image Processing Technology, Data Sources and Software for Commercial and Scientific Applications*, JPL Document JPL SP 43-30, November 1976.
7. Bryant, N. A., "Integration of Socioeconomic Data and Remotely Sensed Imagery for Land Use Applications," *Proceedings of the Caltech/JPL Conference on Image Processing Technology, Data Sources and Software for Commercial and Scientific Applications*, JPL Document JPL SP 43-30, November 1976.
8. Zobrist, A. L., N. A. Bryant, and A. J. Landini, "Use of LANDSAT Imagery for Urban Analysis," *Proceedings of Urban and Regional Information Systems Conference*, Kansas City, Mo., August 1977, in press.
9. Zobrist, A. and B. McLemore, "Computability of Five-Color Maps," *Image Science Mathematics Symposium Proceedings*, Monterey, California, November 1976 (available from Western Periodical Company, North Hollywood, California 91605).

Design of pipelined systems for landsat image processing

by DONALD C. BRABSTON and JOHN E. TABER
TRW Systems
Redondo Beach, California

INTRODUCTION

With two LANDSAT satellites in orbit and a third to be launched in the near future, the use of imagery obtained by satellite for monitoring the earth's resources is well established. The timely, efficient, and routine handling of this image data is not so well established, however, and large systems for this purpose are just now coming into being. Two examples of such systems are NASA's MDP system being developed by IBM, and Department of the Interior EROS Data Center's EDIPS system being developed by TRW. Under consideration for the future is a system for handling imagery from LANDSAT D, whose data volume and throughput requirements are substantially more severe. This paper discusses some of the design considerations forced by the requirements of such systems and suggests solutions.

The main driving requirements for systems processing LANDSAT data are the extremely large data volume and the high throughput necessary to handle the data in a timely manner. These two requirements also separate the design of these systems from normal data handling systems which deal with lower volumes and throughput. The nature of the processing performed on LANDSAT data also differs from the processing performed in other applications. Special processing may include geometric warping of the imagery, radiometric correction, edge enhancement, and manual display interaction. Finally, the output products of these systems include annotated image films, and digital tapes of the imagery for specific users. The data volume and throughput, special processing, and special output products of LANDSAT data all press for unique solutions not found in other data processing systems.

Due to the high throughput and variety of special processing to be performed on the data, a pipelined design for the system is frequently the best choice. With a pipelined system, data from one scene may be read into the system while data from another scene is having special processing performed, and yet another scene is being output to film. Pipelining here results in maximizing system resource utilization and throughput while minimizing the total amount of hardware required. It is thus an efficient, high throughput, and cost effective design. Pipelining at a low level may mean the use of pipelined signal processors or microprocessor hardware. Pipelining at a high level involves the use of several general purpose computers, each of which is dedicated to performing a processing segment in the pipeline flow, in parallel with the other computers. What level of pipelining is best depends on the specific application and the hardware available. Figure 1 shows the processing flow of a typical pipelined system.

The next section discusses the specifics of LANDSAT image systems requirements; the third section briefly discusses special processing performed on the data; the fourth section discusses key design considerations and suggested solutions; and the last section summarizes the key points.

SPECIFIC LANDSAT IMAGE SYSTEM REQUIREMENTS

LANDSAT image processing system requirements are driven by the image sensors aboard the spacecraft.[1] For LANDSAT's 1, 2, there are two sensors: The Multispectral Scanner (MSS) and a set of Return Beam Vidicons (RBV). The vidicons have not been operational since early in the mission. On LANDSAT C there is an MSS and a pair of higher resolution RBV's. The MSS has 26 detectors arrayed to image four spectral bands with a resolution of 79 meters and a fifth band with a resolution of 237 meters. In geometrically correcting the MSS scenes, the images are oversampled so that each picture element (pixel) is 57 meters square. An MSS scene is 185 kilometers square, so that an output image is $4^1/_9$ bands of about 3300^2 pixels. A similar computation yields a LANDSAT C RBV scene of about 5300^2 pixels, but only one band. A complete MSS scene is imaged roughly every 30 seconds, and the sensor is used over one hour per day. Combined data from LANDSAT's 1, 2, and C totals about 200 MSS scenes and 160 RBV scenes per day. Thus, a typical system requirement is to process this data in a two-shift working day. Total data volume is about 14 billion bytes per day. Allowing for gaps, spacecraft telemetry data, and annotation data (added in later processing) on the high density tapes on which the data is typically recorded, the system must be able to process data at a continuous rate of about 500,000 bytes per second.

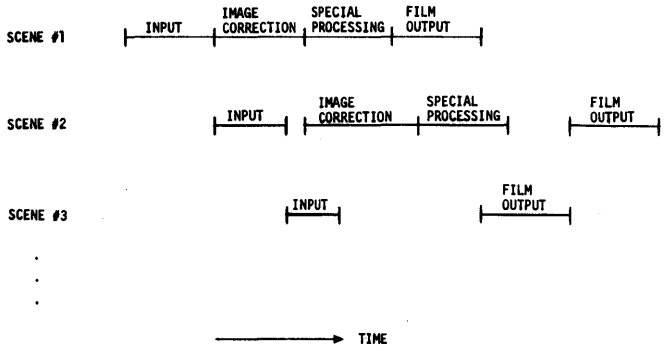


Figure 1—Processing flow of a typical pipelined Landsat image processing system

In addition to throughput requirements, LANDSAT image processing has accuracy requirements on the special geometric and radiometric correction processing performed on the data. Typical requirements are that the output pixels be accurate to within one gray level (eight bit data having 256 gray levels) after radiometric and geometric correction, and that the geometric correction algorithm used introduce no more than $^1/_{10}$ of a pixel spacing error (i.e., $^1/_{10}$ of 57 meters for the earlier LANDSAT's) in the location of the output pixels. Further, in the case of geometrically correcting the data to register it to other LANDSAT images or to control points on the earth with known locations, the position error of output pixels may be required to be less than ⅓ or ½ pixel spacing. These accuracy requirements impose stringent conditions on the algorithms used, more than on the system design. However, the overall system design may well be impacted by the increased complexity and computation time of the sophisticated algorithms necessary to meet the accuracy requirements.

Reliability requirements for this type of system cannot be overlooked in the system design. System availability requirements of 85-90 percent are typical. Hardware redundancy, switchable peripherals, hot spares, and modular replacement of hardware components at the board level are all potential methods for meeting the reliability requirements and must be considered in the selection of a system design. Software reliability is also important. Detailed fail-safe error checking by the software modules, top-down design, and structured programming can and do help to insure reliable software, and are considered in the design and development of the system.

Although requirements on accuracy, reliability, special processing, etc., are important, the throughput requirement remains the most critical in selecting a system design. How these requirements determine the key issues in the system design will be discussed in the fourth section.

SPECIAL PROCESSING

The nature of LANDSAT imagery frequently requires special processing on the image data. This section briefly discusses some of the special processing, including geometric warping, radiometric correction, image enhancement, and format conversion of the input data. Although not all of these processes are required for every system, they are typical of the kinds of processing done on LANDSAT data and frequently drive aspects of the system design.

Geometric correction

The characteristics of the spacecraft, earth, and imaging sensor combine to distort the true ground data into the raw image data received from the spacecraft. This raw data must be corrected to show the true ground picture. The effects of earth rotation, spacecraft position and attitude, sensor scanning nonlinearities, earth curvature, and time delay between adjacent detector's exposures must all be removed by the geometric correction processing. In addition, the output products are frequently required to be in one of several map projections, the most common being the Universal Transverse Mercator (UTM[2]). Finally, the image data may be required to be registered to an earlier scene or to ground control points of known location. These effects are also allowed for in the geometric correction processing.

The geometric correction processing may proceed in two stages (which may be two segments of the pipeline). The first is the generation of a set of values at a grid of regularly spaced locations throughout the corrected image. The values specify the location of the corresponding pixels in the input, uncorrected image. The accuracy requirement drives the number of grid points, which in turn determines the computation and storage load for this portion of the processing. This grid is generated from the equations for the distortion effects mentioned above, the most significant being the spacecraft attitude and the map projection. The equations are complicated and their computation time must be carefully estimated.

Once the grid of locations is determined, the warping itself must be performed. Each pixel of the output image is located by interpolation from the grid of input locations in the input image to a fractional pixel location. The output pixel value is then interpolated from the values of its input pixel neighbors.

The complexity of the warping algorithm and its parametric nature generally preclude its implementation purely in hardware, but the right choice of a hardware-software mix is not always obvious. The generation of the grid of location values in the first stage of the geometric correction involves computation of complex equations and is usually done in a general purpose computer. The nature of these computations, however, does impact the computer selection.

Radiometric correction and image enhancement

For the MSS sensor (and the TM sensor to be on LANDSAT D) there are multiple detectors with different characteristics. Therefore, different detectors will give different outputs for the same input radiances. Removing the detector

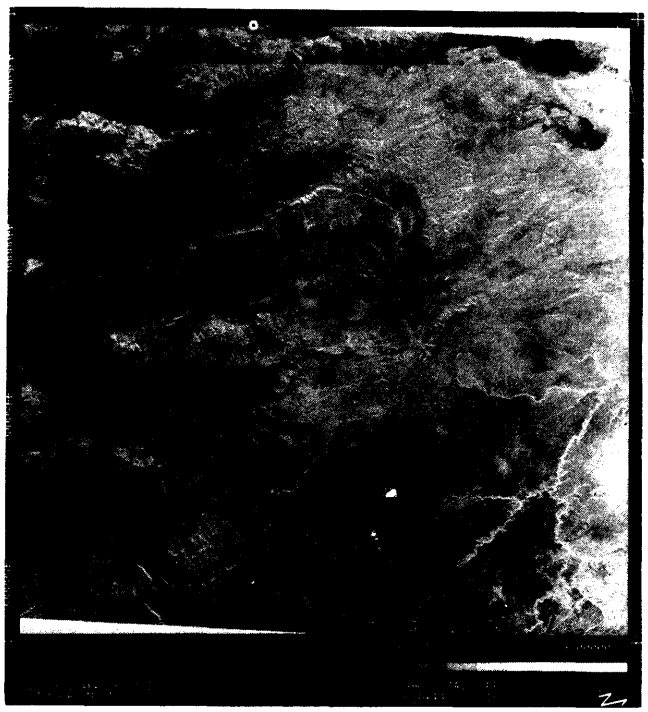
specific errors in the pixel values is known as radiometric correction. This may be done by constructing a table for each detector which maps the pixel values received from the spacecraft into radiometrically correct values. Thus, for LANDSAT's 1, 2, and C there would be 26 tables of 64 elements each (since the data as received is 6 bits per pixel). As with geometric correction, this proceeds in two stages (which again, may be separate segments of the pipline). The first stage is the construction of the detector-specific tables themselves, and the second is the mapping of the pixels through the tables. Construction of the tables may require obtaining statistics about the image data in the form of mean and variance or histograms of the pixel values or calibration data in order to remove image striping caused by the differences between detectors. In a general purpose computer, computation of these statistics requires several microseconds per pixel; hence, special purpose hardware or firmware is frequently dictated by the requirement to construct these tables.

Once the tables are constructed, the pixels must be mapped through the tables to generate the radiometrically corrected values. Since this again requires handling each pixel, special purpose hardware may be the solution to mapping the pixels at the required throughput.

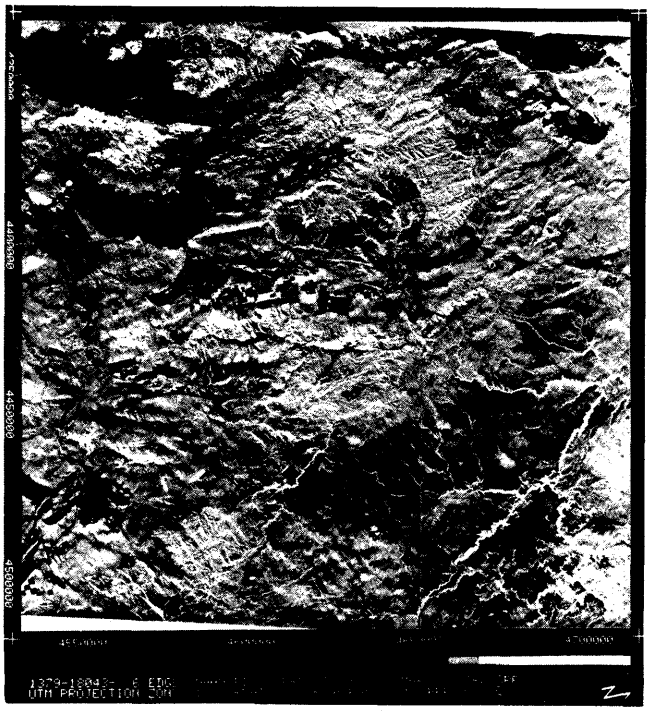
Image enhancement may involve filtering the input (corrected or uncorrected) image to enhance edges, or remove haze or contrast stretching the image so that scenes of low contrast look better visually. These processes may take advantage of the scene statistics gathered at an earlier stage of the pipeline. Depending on the complexity of the algorithms used, special purpose hardware or firmware may be used, but again, normally a general purpose computer lacks sufficient speed to handle these operations. Figure 2 shows a corrected unenhanced image compared to the same image after edge enhancement.

Format conversion

LANDSAT image data as received from the satellite is interleaved by detector in a formal called "band-interleaved-by-pixel" (BIP), i.e., one pixel is received from each detector before the second pixel is received from each detector. Since an image line is formed by one detector as it scans the earth, in order to reform the image into image lines, the data format must be converted first into lines (band-interleaved-by-line or BIL) and finally into band-sequential (BSQ) where all lines of an MSS (or TM) spectral band are consecutive. Converting from BIP to BIL again involves handling each pixel, but only moving it from one location to another. Thus, specially addressable memories or computer controlled direct memory access devices (DMA's) may be the solution here. For 26 detectors of 2300 pixels per input line, a memory of 60,000 bytes would be required—quite reasonable for modern minicomputers. Conversion from BIL to BSQ, however, could require a memory large enough to hold an entire scene (31 million bytes for the earlier LANDSAT's) unless enough devices were available in the next pipeline segment to handle all bands simultaneously. Thus, a mass storage device such as a disk is usually needed



UNENHANCED LANDSAT SCENE



EDGE-ENHANCED LANDSAT SCENE

Figure 2

here. Formatting this storage device so that image data is written on in BIL and later read off in BSQ, both at the required data rates, is a nontrivial task. Disk formatting will be discussed in more detail in the next section.

The special processing described in this section often necessitates special purpose hardware or firmware for handling the individual pixels. Even where pixels are not handled directly, the algorithms involved may impact hardware selection. The next section discusses further these and other key design issues.

KEY DESIGN ISSUES

The requirements of throughput, accuracy, reliability, economics, and special processing described above, force the selection of a system design. This is not to say that there is only one solution to these problems, but that there are certain key design issues which must be dealt with. Some of these, such as the choice of special hardware and memory sizing have been mentioned above. Other issues are the selection of general purpose computers, the need for a process control scheduler to handle the pipeline flow, efficient use of mass storage devices, and software development methodology. Obviously all of these cannot be discussed in depth here due to lack of space; however, this section looks briefly at each issue, showing the problem, and suggesting a method or methods for its solution.

System loading considerations

The throughput requirement of hundreds of kilobytes per second frequently means many transfers per second through a general purpose computer's memory, perhaps several times the overall throughput rate. For example, suppose the system of Figure 1 were to be implemented on one general purpose computer. The transfers shown in Figure 3 may then be executed for each scene. Although the pipeline segments' processing occurs sequentially for a given scene, normally the pipeline will be kept full and all of these segments will be executing on different scenes simultaneously.

In this example, ten transfers to and from memory are executed per pixel. At a data rate of, say 700,000 bytes per second on the average, this means an average data rate of

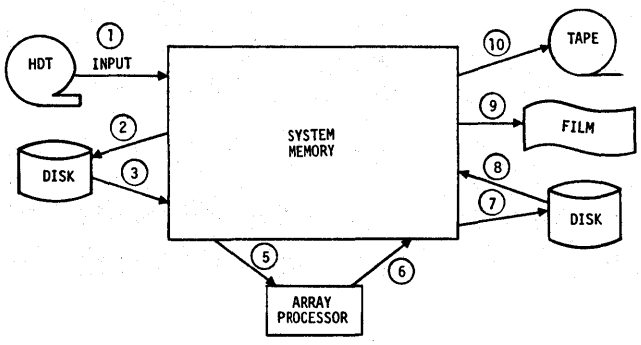


Figure 3—Memory transfers in a pipelined system

7 million bytes per second. In practice, the actual peak data rate may be substantially higher than this. For example, although the high density tape (HDT), film generator, and array processor transfer at a fairly constant rate, the disk and tape transfers occur much faster while actually performing the record transfers than the average rate which includes inter-record gaps. Thus, the transfers 2, 3, 7, 8, and 10 may reach 1.2 million bytes per second each. This gives a total peak rate of 9.5 million bytes per second (or actually more since the CPU normally fetches instructions and data from this memory as well). Thus, system loading considerations may force the division of the flow into specific pipeline segments independent of the allocation of processing functions to segments. This requires a memory with an access time of about 100 nanoseconds per byte. Furthermore, in the case of many minicomputers (e.g., PDP-11, ECLIPSE, SEL 32, etc.), the data must travel via one memory bus; hence, this bus must be capable of sustaining the data rates calculated here.

One potential solution to this problem is the selection of a computer with several data busses or even high speed I/O channels dedicated to transfers between certain peripherals and memory. Still another solution might be the selection of several banks of memory, each with its own memory controller to give independent parallel access to each bank. With this solution, (assuming the memory bus or busses could handle the data rate) buffers could be placed in different memory banks to reduce the load on each individual bank to an acceptable level. A variation on this solution would be to interleave the memory banks with a least-significant-bit addressing scheme to reduce the effective access time of the memory. This has a particular advantage here since most transfers are of image lines which transfer from successive locations in memory. A final solution might be to construct the process control software described later to preclude processes which strain the system's memory or bus capacity from running in parallel. If all of these potential solutions fail the analysis, more computers may be used, even to the ultimate extent of having one computer per pipeline segment. (Unfortunately, this may dictate still another computer to provide process control for the pipeline flow.)

A further problem occurs in considering the computational load on the computer's central processing unit (CPU). This is related directly to the software used by the system.

Current minicomputer operating systems, for example, have software overhead times of ½ millisecond per interrupt. If the system of Figure 3 receives an interrupt on completion of the transfer of every image line, there will be an interrupt about every 4 milliseconds for each of the ten transfers. This allows .4 msec for processing each interrupt, which may well be beyond the capacity of the system. Potential solutions to this are to get a new, faster operating system or to write device handlers which speed the interrupt processing or even trigger the start of the next transfer when the current one is complete. Another measure would be to generate an interrupt only after the transfer of several lines. Whether this is possible depends on the image data formats and the peripheral and computer hardware.

Hardware/software/firmware tradeoffs

The variety of special processing being performed on image data requires a mixture of general purpose computers and their software, firmware in the form of microprocessors or array processors, and special purpose hardware. In nearly all large scale, pipelined systems there will exist a need for a process controller. Due to the complexity of this scheduler and the need to control the various peripherals, a general purpose computer will normally form the heart of the system. This computer may also be well-suited for interacting with operators and users via displays or terminals. In addition, it may perform some complex, low throughput special processing such as the generation of the grid of geometric correction locations or the radiometric correction tables from the scene statistics described earlier.

However, a general purpose computer is not normally suited for processing that deals with pixels on an individual basis, due to a lack of the speed required by such processing. Thus, either special firmware of hardware must be employed. The choice of firmware or hardware depends primarily on the complexity of the processing and the throughput required. As an example, the more complex functions of geometric warping and edge enhancement would probably be done in firmware if the required throughput could be met, while the format conversion and contrast stretching could be done in hardware. The gathering of scene statistics and radiometric correction could be done in either hardware or firmware, depending on cost effectiveness. Very complex algorithms such as geometric correction could be done in commercially available array processors (such as Floating Point Systems AP-120B, Signal Processing Systems SPS-81, or CSP, Inc.'s MAP-300). If necessary to meet the throughput requirements, an image can be split into parallel streams, each passing through a separate array processor with the output being merged again (if the algorithm allows such a split). These array processors are sufficiently complex that configuring them with the proper memory and most suitable options raises design issues on its own.

The decision to use special hardware or firmware in various parts of the system may also force segmentation of the pipeline at specific points. This and cost-effectiveness considerations make this a key design issue.

Process control

One disadvantage of a pipeline system design is the need for a more complex process control program or scheduler. On a strictly sequential system resource conflicts do not usually occur since only one process is in progress at any time. Similarly, a totally parallel system in which all resources are duplicated does not normally give resource conflicts. However, in even a simple pipelined system such as that of Figure 1, resource conflicts do occur, and the scheduling of the pipeline processing segments must be done. In Figure 1, for example, several resource conflicts are apparent: Scene 2 cannot begin its image corrections immediately after input because the array processor required for it is in use for Scene 1; similarly, conflicts arise in the case of the output film generation hardware. Other conflicts may arise not in the use of specific hardware, but in the total system load. If several special processing segments had the hardware to execute simultaneously, they might still exceed the load capacity of the system as far as memory or bus loads are concerned. Further, the scheduler may need to prevent other processes from executing concurrently because of conflicts in accessing shared data areas. In short, in a pipelined system a scheduler may be necessary to eliminate resource conflicts. (If the processing flow is static and the same for all scenes, the scheduler may be very simple or even nonexistent in that the individual pipeline segments tell each other when they are complete or available; but for more complex systems, the scheduler may become quite involved.)

The main functions of a scheduler in this type of system are: to determine the processing flow for each scene, initiate processes, determine processing conflicts and resolve them, and to monitor the processes for termination. All processes can communicate with the scheduler to indicate termination or error conditions, and in general need not communicate directly with other processes. Other scheduler functions may include operator interaction to determine each scene's processing flow and to report error conditions. If the system runs out of resources, the scheduler may stop the flow altogether as the pipeline "backs up." For example, in Figure 1, if the film output generator fails, later scenes will be unable to complete their processing which means that they must be stored on mass storage devices. As these devices fill up, the image correction, special processing, and input segments have no place to store their data, so that finally input must stop. If the input is from tape, data may not be lost; however, if the input is directly from the satellite, data may be lost at the input end of the system. (The system designers could, of course, construct the scheduler so that data is not output to film, but is saved on tape, or other measures to prevent the loss of data.)

The internal design of a scheduler is beyond the scope of this paper, but a scheduler is one solution to the problem of process control, which is an important design consideration.

Use of mass storage devices

In order to keep the process running smoothly, it is frequently necessary to use mass storage devices such as disks as buffers between pipeline segments. Otherwise, lack of a needed resource for even a brief period may bring all processing to a halt as main memory buffers fill up rapidly. In addition, disks may be necessary to store image data written in one order to be read in another order. Since industry standard 80- and 300-million byte moving-head disks are commercially available, these are good choices for mass storage buffers. Several issues arise in using these disks efficiently.

The data formatting on the disk is a design tradeoff which must be made. Figure 4 shows a typical disk format for buffering MSS data. Each track of the disk is hardwired as

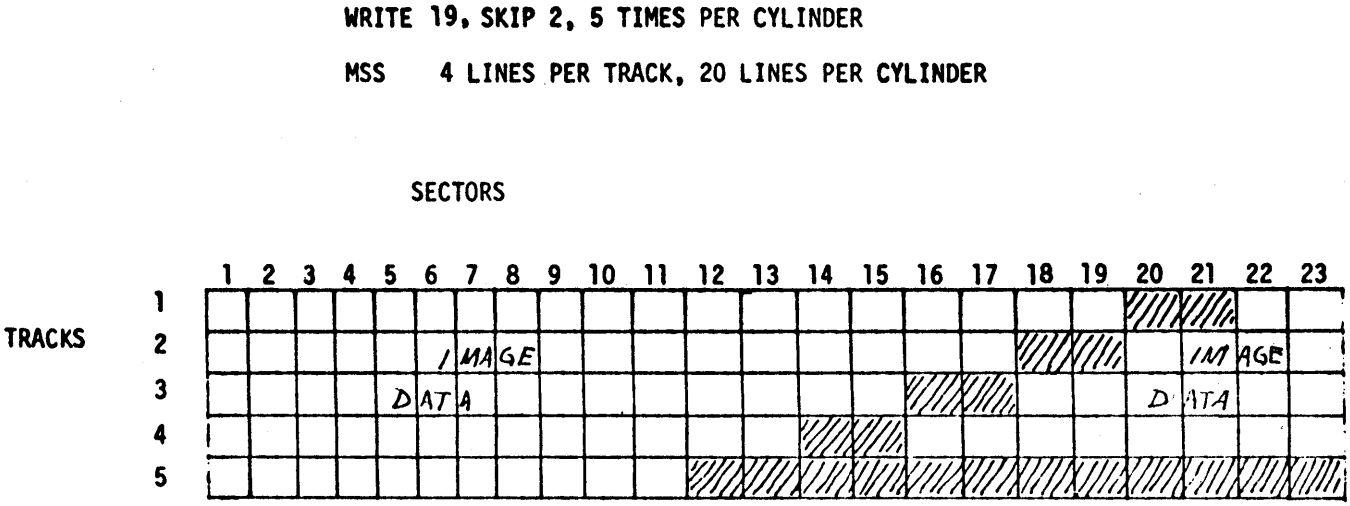


Figure 4—Typical disk format for MSS scene

23 sectors of 768 bytes each. There are 5 tracks per cylinder. There is no time penalty for disk accesses which change tracks within a cylinder, but there is a substantial penalty for switching cylinders which requires a movement of the read/write head. If each image line is 3500 bytes, then 4 lines will fit into 19 sectors. Since the disk rotates at 3600 rpm, it takes about 14 msec to read 4 lines of data. Then 2 sectors are skipped to allow about 1½ msec for software overhead before the next 4 lines are read. Twelve sectors are left empty at the end of the cylinder to allow time for the head movement to switch to the next cylinder. Thus, the reading of 20 image lines requires on the average of 5 disk revolutions. This gives a sustained throughput capacity of 840,000 bytes per second, which is in excess of the required 700,000. Hence, there is some spare time available for situations in which system loading prevents being able to read the next 4 lines within 1½ msec after the last 4 are completed.

Because of the high speed involved, there is no time available for retries of disk reads or writes in the event of parity errors. However, with modern disks, the parity error rate is so low that the expected number of errors affects the image significantly. Thus, parity checking need not be performed. If parity checking is performed, retries should not be made, and a system alarm should be generated only if the error rate becomes too high. Parity checking, formatting, high speed software device handlers, and reliability are some of the key design issues in using mass storage devices efficiently.

Software methodology

In designing high throughput, reliable systems, the hardware design is not the only consideration. The software design and implementation for general purpose computers and for microprogrammable microprocessors and array processors is also extremely important. Software design problems arise in attaining the high throughput required, and in developing reliable software which is free of timing-dependent bugs. Further, as research progresses, new algorithms are developed, and new user needs become apparent, the requirements for LANDSAT image processing change. Therefore, the software should be modular enough to permit easy modification or inclusion of new processing. The scheduler should be sufficiently flexible that addition of new segments in the pipeline does not cause major redesign or recoding of the scheduler.

One means of attaining the high throughput required in array processors is to actually code and benchmark the critical high speed inner loops of the algorithms in the design phase, rather than waiting until the coding phase. This way, a good estimate of the actual throughput of these processing segments can be obtained early.

In developing reliable software, the top-down design process is very valuable. Software tools which aid this process are very beneficial in easily developing a top-level design and expanding it to deeper and deeper levels until the software can be coded directly from the design document. If done properly, the detailed design document also becomes the main software documentation. Two good tools for software design are structured flow charts and Program Design Language (PDL). Impressive cost savings in software development through the use of PDL have been cited.[3] Software design tools aimed at aiding pipelined systems design would be of great benefit. These tools would facilitate checking for deadlocks, use of common data areas, etc. Unfortunately, we know of no such tools available at the present time.

Structured programming is also of benefit in developing software for pipelined systems, and especially for modifying it in the event of changing requirements. In using FORTRAN for the software development, the use of a structured FORTRAN preprocessor can be of benefit, especially when the design was done in a top-down structured fashion.

SUMMARY

This paper has shown how the requirements of large scale LANDSAT image processing systems force the consideration of certain key design issues. The main requirements of the system are the large data volume and high throughput, reliability, need for special processing, and accuracy. Processing peculiar to the LANDSAT systems includes geometric and radiometric correction, image enhancement, and high speed data reformatting. A pipelined design is often a natural choice for these systems. Although there are many issues to be tackled in the design of such systems, only the most important ones have been addressed here. These include system loading considerations in selecting computer hardware; the decision on where to use special hardware or array processors; the need for process control to schedule the pipeline's segments; the efficient use of mass storage devices; and software tools and techniques.

REFERENCES

1. *LANDSAT Data Users Handbook,* NASA, Goddard Space Flight Center, 1976, Document 76SDSA258.
2. Deetz, Charles H. and Oscar S. Adams, *Elements of Map Projection,* U.S. Government Printing Office, 1945.
3. Caine, Stephen H. and E. Kent Gordon, "PDL—A Tool for Software Design," *Proceedings National Computer Conference, 1975.*

AOIPS—An interactive image processing system

by P. A. BRACKEN, J. T. DALTON, J. J. QUANN and J. B. BILLINGSLEY
Goddard Space Flight Center
Greenbelt, Maryland

INTRODUCTION

Prior to the fall of 1976, computer aided data analysis support for applications investigations at NASA's Goddard Space Flight Center was conducted on large scale IBM System/360 computers in a batch oriented environment without the aid of image display units. As a result, even simple image processing operations took from days to weeks to produce desired results and required numerous manual, errorprone steps to produce the desired end product. In addition, operations which required a high degree of human interaction with the image data during the information extraction process (e.g., derivation of wind fields from cloud motions in a time-lapsed series of satellite images) were not practical in the batch environment.

The Atmospheric and Oceanographic Information Processing System (AOIPS) was developed to help applications investigators perform required interactive image data analysis processes rapidly and to eliminate the inefficiencies and problems encountered with the batch operation discussed above. The major AOIPS hardware components are shown in Figure 1.

Initial hardware deliveries occurred in September 1975 and consisted of a PDP-11/70 computer and peripherals and a modified General Electric Company Image 100 system, which included an interactive image analysis terminal and a PDP-11/45 computer. The Image 100 system was designated as AOIPS Image Analysis Terminal 1 (IATl). Two additional image analysis terminals, designed at Goddard Space Flight Center and implemented by Hazeltine Corporation, were delivered in May of 1976 and May of 1977. These terminals have been designated as AOIPS Image Analysis Terminals 2 (IAT2) and 3 (IAT3).

In parallel with the hardware development activities, a series of applications software packages was developed to implement specific information extraction processes to meet established image data analysis requirements. AOIPS operations during 1976 and 1977 have demonstrated its capability to solve many operational image processing problems, and has greatly extended an investigator's ability to perform image information extraction operations in a flexible, efficient manner.

SYSTEM OVERVIEW

The heart of the AOIPS is a Digital Equipment Corporation PDP-11/70 computer to which three magnetic tape drives, three disk units and other standard peripherals are attached. The image analysis terminals interfaced to the PDP-11/70 central processing unit provide for user interaction in the extraction of information from digital images.

Terminal 1 (see Figure 1) provides capabilities for time lapsed display of image data and for output of video signals to a video disk storage unit. Control of this terminal is provided by its own PDP-11/45 computer. Digital images on computer-compatible tapes are loaded into a 5-channel solid-state refresh memory from two 800/1600 bit-per-inch (bpi), 9-track tape drives. The contents of the refresh memory channels are input to analysis console digital logic for ratioing, scaling, and limit comparison; the output of these operations is displayed on both color and black and white TV monitors. Using the analysis console logic, image enhancement and multispectral classification analyses are performed under control of software on the PDP-11/45. Terminal 1 is interfaced to the AOIPS PDP-11/70 computer through a dual port, shared-disk unit capable of formatted storage of 88 megabytes (8 bits per byte) of data.

Terminals 2 (see Figure 1) and 3 are identical and are interfaced directly to the PDP-11/70 through a high-speed direct memory access interface for control signals. Using the three 800/1600 bpi, 9-track tape drives of the PDP-11/70 and the High Density Digital Tape (HDDT) unit, magnetic tapes are loaded into the 5-channel solid-state refresh memories of Terminals 2 and 3. Images may also be loaded from disk or from maps or photographs digitized by a video input scanner interfaced to the terminal. The contents of the refresh memory channels pass through lookup tables, digital matrix switches, and other image manipulation logic in the console. Image enhancement and parameter extraction analyses are performed under control of software on the PDP-11/70.

The video disk, which is connected to all three terminals through a video switching system, has the capacity to store up to 600 TV-sized images. Using the switching system, images may be recorded from or played back to the TV monitors of any of the terminals.

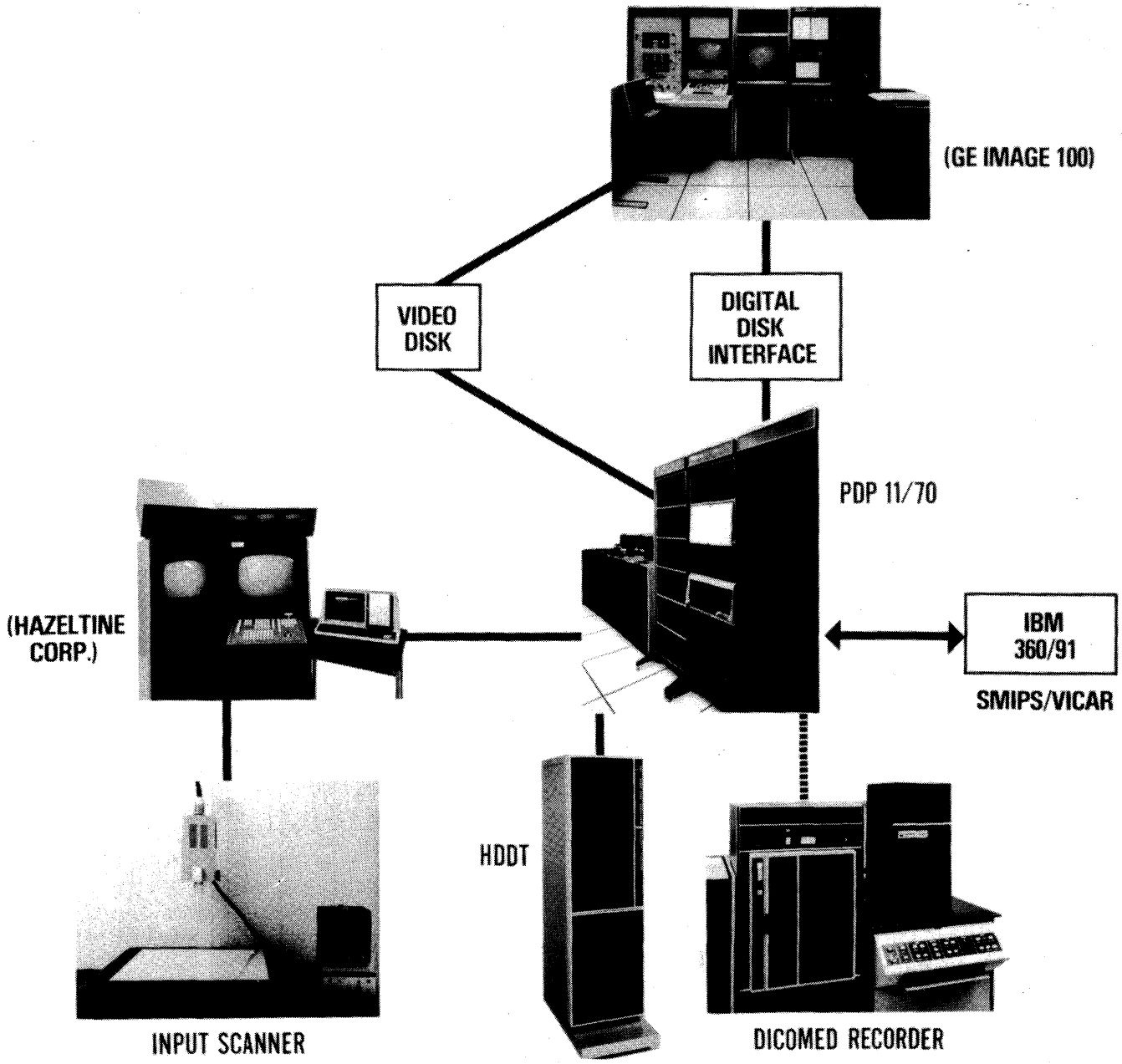


Figure 1—Atmospheric and Oceanographic Information Processing System—AOIPS

The AOIPS PDP-11/70 computer communicates with a remotely located IBM S/360/91 computer through a 4800-bit-per-second telephone line connection.

A Dicomed Image Recorder (shown in Figure 1) functions as an off-line device to convert information recorded on computer-compatible tapes (CCTs) into photo products of image and related ancillary data.

AOIPS utilizes computer-compatible tapes of multispectral digital image data generated by satellite and aircraft image scanners as well as related digital ancillary data required to perform specific information extraction operations. In the future, high density digital tapes will provide the primary input medium for meteorological satellite data. The system provides capabilities for inputting digital data in several formats; performing standard image preprocessing operations including image registration, geometric correction, scanner distortion correction, and zooming and reducing subimages; executing image processing functions including level slicing, contrast enhancement, pseudo color and false color combinations, band ratioing, linear combinations, and histogram generation; performing analytical operations including multispectral classification and special applications

analyses; and displaying and generating hard copy output of single images overlayed with data plots, contours and various forms of image annotation.

USER—SYSTEM INTERFACE

There are two types of users supported on the AOIPS: (1) applications investigators who specify and perform information extraction operations on the system for specific applications, and (2) programmer/analysts who develop new systems and applications software. Applications investigators interface with the system by requesting execution of various system processes from menus presented on interactive cathode ray tube display units. Adopting the menu approach as the primary user interface has simplified training problems and virtually eliminated the requirement for a user to learn a new computer language. A discipline oriented user is led through the entire sequence of possible processing options available by a series of prompts and a layered structure of menus which require simple responses.

Typical user data analysis sessions last for about two hours. During a session, the user has several options available to him for displaying intermediate results, generating output products, and defining sequences of processing operations. Operations may be terminated with the capability to restart in subsequent analysis sessions on the system. The user has the option to start and to save/restart analysis sessions using magnetic tape and/or disk units. The disk operations provide a significant speed advantage over the magnetic tape operations.

The second type of system user, the programmer/analyst, typically utilizes operating system facilities and the operating system commands to code, compile, link, edit, and checkout the software being developed. The programmer/analyst does not usually interact in the menu environment until the final checkout and integration phase in the development of a new applications software package.

Capabilities are being developed to allow users to define and automatically execute a sequence of menu processing steps in one interactive operation by specifying a command string which is used to initiate and control the desired processing sequence.

HARDWARE DESCRIPTION

A more detailed version of the AOIPS hardware block diagram is presented in Figure 2.

Figure 2 indicates which devices are interfaced to the PDP-11/70 through high-speed direct memory access (DMA) channels and which devices are interfaced through the UNIBUS input/output channel. The interface between IAT2, IAT3, and the PDP-11/70 is a dual interface in which all high-volume data transfers use a high-speed channel and all low data volume, control signal information transfers use a UNIBUS channel interface.

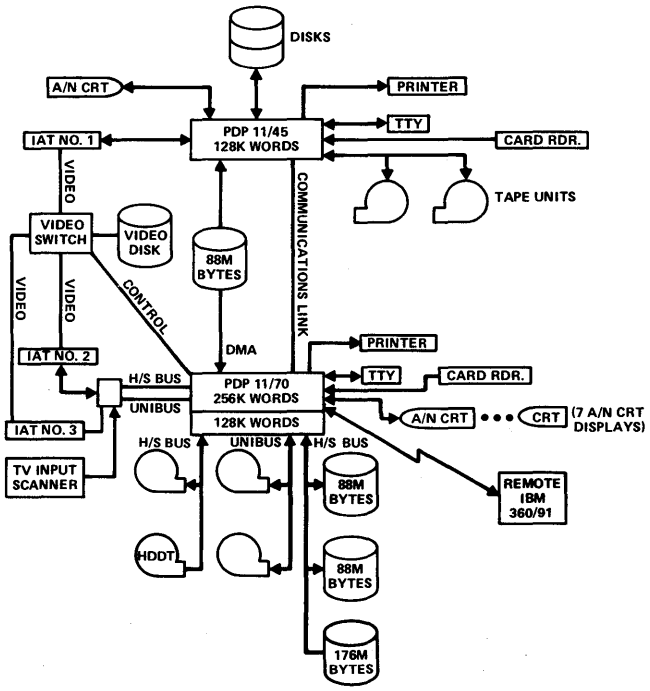


Figure 2—AOIPS configuration

Figure 2 indicates the functional interconnections of the video switching system incorporated into the AOIPS configuration. All video signals from the three Image Analysis Terminals are routed to the video switching unit. The video switch itself is under computer control by the PDP-11/70. By utilizing this control link, any video signal within the system can be routed to the video disk for storage, to video recorders, and to a large Advent TV projection screen or other TV monitors interfaced to the switching unit. It is also possible to display video which originates in any one of the terminals on any other terminal. This capability is utilized for applications requiring simultaneous use of multiple image analysis terminals during information extraction operations. In addition, the video disk unit is used to store sequences of up to 600 TV sized (512 lines by 512 picture elements per line) images for time-lapsed playback through the system.

Figure 2 also indicates the 9600-bit-per-second back-to-back DL11 teletype interface link between the PDP-11/45 and 11/70 computers. This link is used primarily to communicate synchronization and control information between the two central processing units during shared disk operations.

The shared disk unit is used to transfer data between the image analysis terminals, to transfer results of a TV input scanner operation to Terminal 1, and to provide a medium for sharing peripheral equipment between the PDP-11/70 and 11/45 computers.

The TV input scanner scans image data into any user selected channel of one of the refresh memories of either Terminal 2 or 3. The contents of this refresh channel can be rerouted to any storage device or any image display unit in the system.

One of the 88-megabyte disk units shown in Figure 2 is dedicated to Terminal 2 users for save/restart operations, the second 88-megabyte disk is dedicated to Terminal 3 users, and the 176-megabyte disk is used to support system functions and software development activities. In addition, users on either of the PDP-11/70 Image Analysis Terminals will be able to select from one to ten refresh memory channels for their image processing operations.

The PDP-11/70-S/360/91 computer link shown in Figure 2 consists of a 4800-bit-per-second remote job entry communications link. This communications path will be upgraded to a 50-kilobit-per-second link in early 1978. It will provide capabilities for AOIPS users to perform large computational jobs on the high-speed IBM S/360/91 computer and to display and analyze the results of these computations on the AOIPS. The kinds of information extraction tasks requiring this support include multispectral classification of large subimages or full images and a variety of weather, climate, and earth resources modeling activities.

Experience in interfacing various kinds of equipment to the AOIPS PDP-11/70 computer indicates that from 50 percent to 70 percent of the total theoretical system input/output bandwidth (5.8 megabytes per second) can be achieved in the current configuration. Figure 3 is an estimate of the system input/output bandwidth achievable for various levels of system activity within the current AOIPS configuration.

Experience with the AOIPS 88-megabyte disk units indicates that 60 percent to 70 percent of the theoretical input/output bandwidth (675,8400 bytes per second) can be achieved in the current system configuration.

The addition of Terminal 3 resulted in approximately 80 percent of the achievable system bandwidth being used during planned peak load conditions. For this reason, no additional image analysis terminals will be attached to the existing AOIPS PDP-11/70 computer in the future.

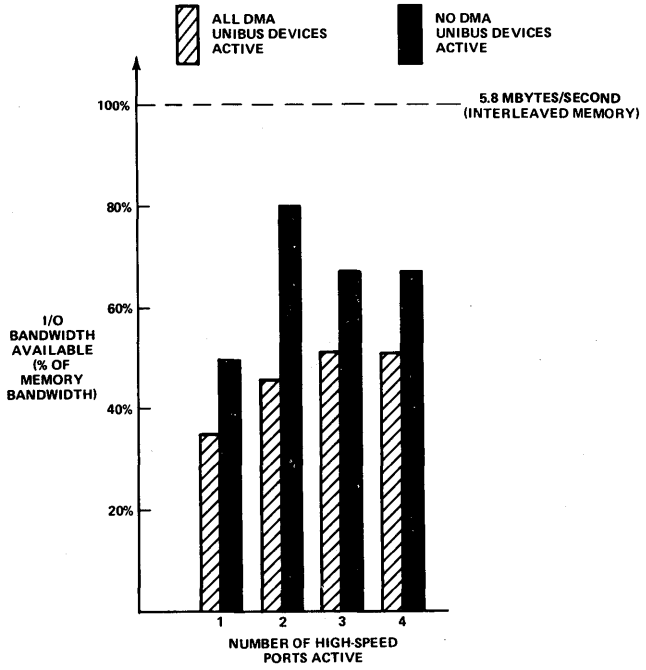


Figure 3—AOIPS I/O bandwidth vs. system activity

Further information about the alternative hardware configurations and peak load conditions considered for AOIPS is presented in Reference 1. Additional information about the existing system capabilities and a complete listing/description of the system hardware and software is contained in Reference 2.

Image analysis Terminal 1

The hardware configuration for the AOIPS PDP-11/45 Terminal 1 subsystem is shown in Figure 4.

The Image 100 Image Analyzer Console contains special-purpose hardware which operates in conjunction with applications software to rapidly analyze multispectral image data.

The Image 100 console processing hardware is designed to process image data, particularly Landsat satellite data, and to perform a parallelepiped classification of multispectral data. The user interacts with the Image Analyzer Console to execute processing functions, to select training sites, and to manipulate themes generated as a result of a classification operation.

The Image 100 Image Analyzer Console performs the following functions in hardware:

- Image Correlation—Two images are displayed alternately in a flicker presentation for image comparison and registration operations.
- Image Scaling—Thumbwheel switches are used to perform individual amplitude scaling shifts on each refresh memory channel.
- Ratioing—Images stored in the refresh memories may be ratioed and the results of the ratio operation displayed on the TV display unit.
- Image Transformation—General purpose transformation hardware exists to rotate feature axes in spectral space.
- Spectral Signature Acquisition—Hardware exists to compile histograms (frequency versus digital gray level) of areas specified by the user in each refresh memory channel. From these histograms, upper and lower gray level limits are obtained to serve as a four-dimensional parallelepiped describing the spectral signature of the specified area.
- Cursor Control—A multimode hardware cursor is sized and positioned by joystick control for the definition and selection of subimage areas.
- Image Analyzer—Special-purpose digital logic operates under software control to store and test for upper and lower gray level limits in each refresh memory channel. Parallelepiped signature regions are synthesized into decision boundaries by the Image Analyzer for classification operations.
- Theme Synthesizer—Circuitry is included in the Image Analyzer to combine themes using logical AND, OR, Exclusive OR, and theme inversion operations.

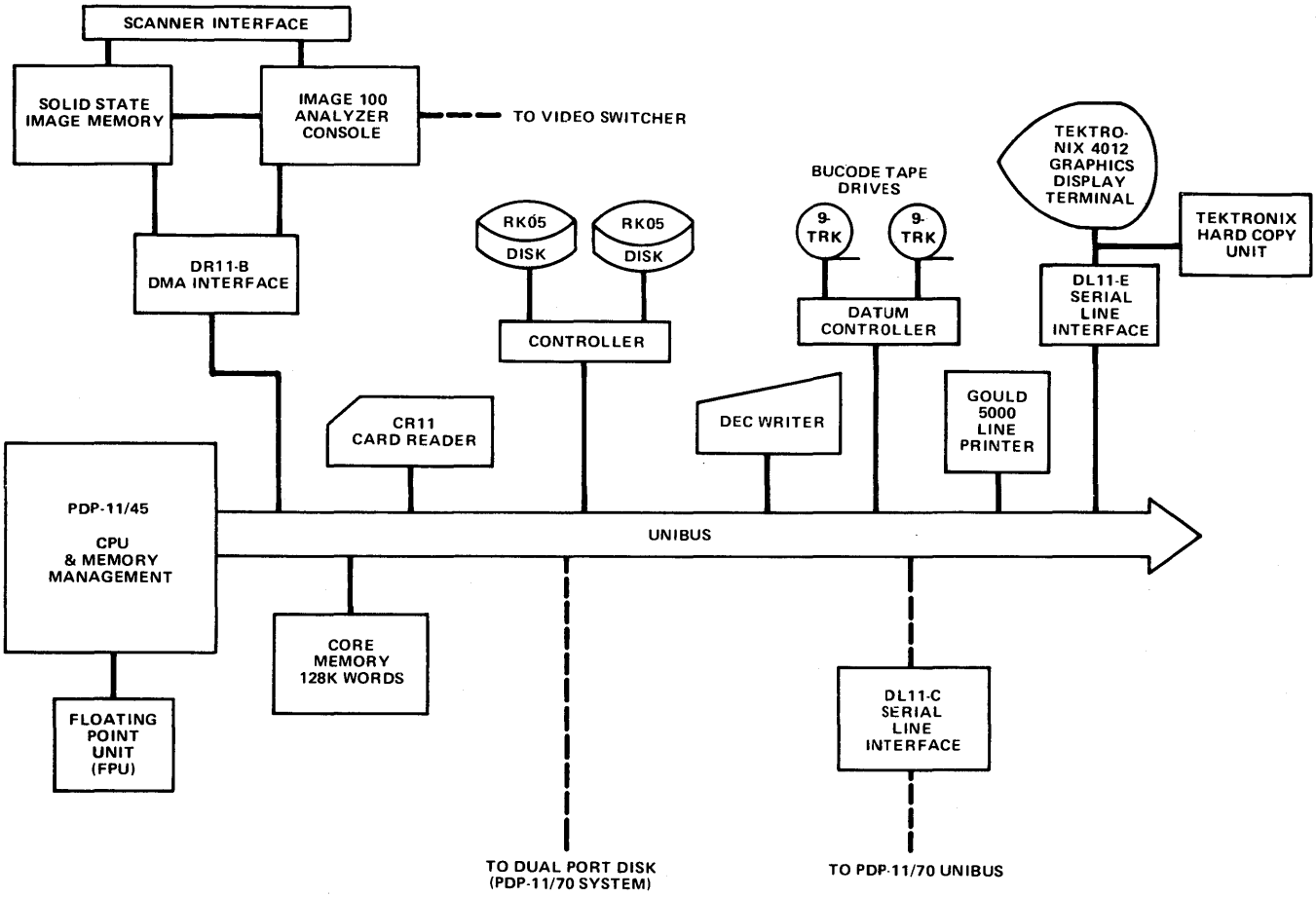


Figure 4—AOIPS PDP-11/45—Terminal 1 configuration

- Color TV Display—A color TV display provides capabilities to display image data, the results of various processing functions and thematic results of classification operations. Software also exists to generate split screen presentations.

Terminal 1 also incorporates hardware modifications for transferring video signals to and from the AOIPS video switching unit.

The two RK05 disk units shown in Figure 4 are used primarily for program storage. The dual ported RP04 disk shared with the PDP-11/70 is used for data storage and for transferring data to and from peripherals attached to the PDP-11/70 computer. The back-to-back DL11-C Serial Line Interfaces shown in Figure 4 (and in Figure 5) are used by the shared disk software to transmit synchronization and control information between the PDP-11/45 and 11/70 processors.

Image analysis Terminals 2 and 3

The hardware configuration for the AOIPS PDP-11/70—Terminal 2/3 subsystem is delineated in Figure 5. Terminals 2 and 3 provide significant image processing capabilities that are not available in second generation terminals currently being marketed. Figure 1 depicts the terminal analyzer console as seen by the terminal user. The terminal was designed to register, process, display, and analyze digital image data for a wide variety of applications. It rapidly performs many image manipulation functions in special purpose hardware which is set up and controlled by the AOIPS PDP-11/70 computer.

Computer control of the terminal hardware allows applications system designers to include hardware control functions in software designs and provides the capability to utilize the full power of the terminal to meet image processing requirements in a highly flexible and responsive system environment.

Figure 6 presents a block diagram of Terminal 2 logic components. Noteworthy elements in Figure 6 include the hardware lookup tables, the digital matrix switches, a light pen, the programmable function button array, and the five solid state refresh memories. Refresh memory 5 of both terminals is randomly addressable to the bit, byte (8 bits) or word (16 bits) level. Each of the refresh memories 1 through 4 are randomly addressable in groups of 16 words.

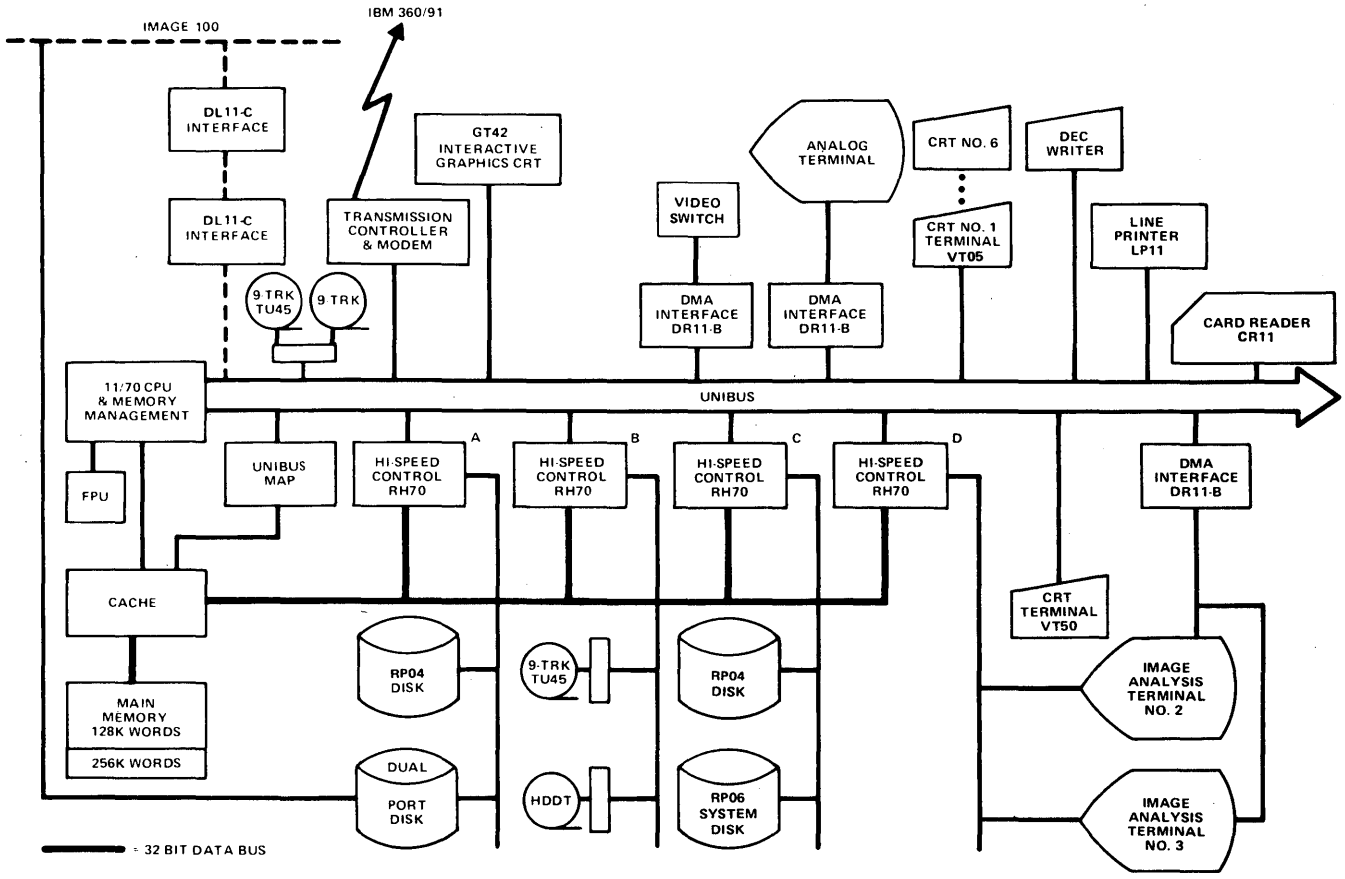


Figure 5—AOIPS PDP-11/70—Terminal 2 configuration

Terminals 2 and 3 provide hardware capabilities for:

- Image Display—The contents of each refresh memory channel are displayed on individual black and white TV monitors. Any refresh channel can also be displayed on a high quality, twenty-one inch black and white monitor. False color displays combining any three refresh channels as well as pseudo color presentations of individual channels are displayed on a nineteen inch color TV monitor.
- Time Lapse Display—The terminals provide time lapsed display of up to ten TV sized images in a movie loop presentation.
- Image Correlation—A hardware fade control is used for simultaneous display of two images during image registration and correlation operations.
- Image Translation—Hardware capability is provided for X-Y translation of images within the refresh memories under joystick control for rapid image registration.
- Graphic Overlay—Graphic overlay of image data sets with a cursor, a rectangle, randomly shaped polygons, plots and data contours is provided.
- Joystick Control—Computer control of a hardware joystick is provided to select the size and position of graphic overlays including a multimode hardware cursor, to select single picture elements or entire sub-images, to control the direction and speed of image scrolling operations, to fly one image over another for rapid image registration, and to control the speed of dynamic color table lookup 'rolling' operations.
- Image Scrolling—Flight line display of image data sets is provided under joystick control.
- Split Screen Displays—Simultaneous multiple image split screen displays are provided.
- Lookup Tables—Hardware lookup tables are used for dynamic color rolling for continuous changing of gray level—color assignments, radiometric correction, parallelepiped classification (up to 9 spectral channels and 8 classes), level slicing and contrast stretching. Each of these operations is performed within $^1/_{30}$th of a second.
- Light Pen Detection—A light pen is used to outline and select irregularly shaped image areas for processing.
- Signature Acquisition—A series of hardware registers is used to rapidly generate gray level histograms of image areas for spectral signature acquisition.
- Theme Synthesis—Thematic information is synthesized using the lookup tables and matrix switches.
- Image Combination—Images in individual refresh channels can be combined and re-recorded into the refresh memory.

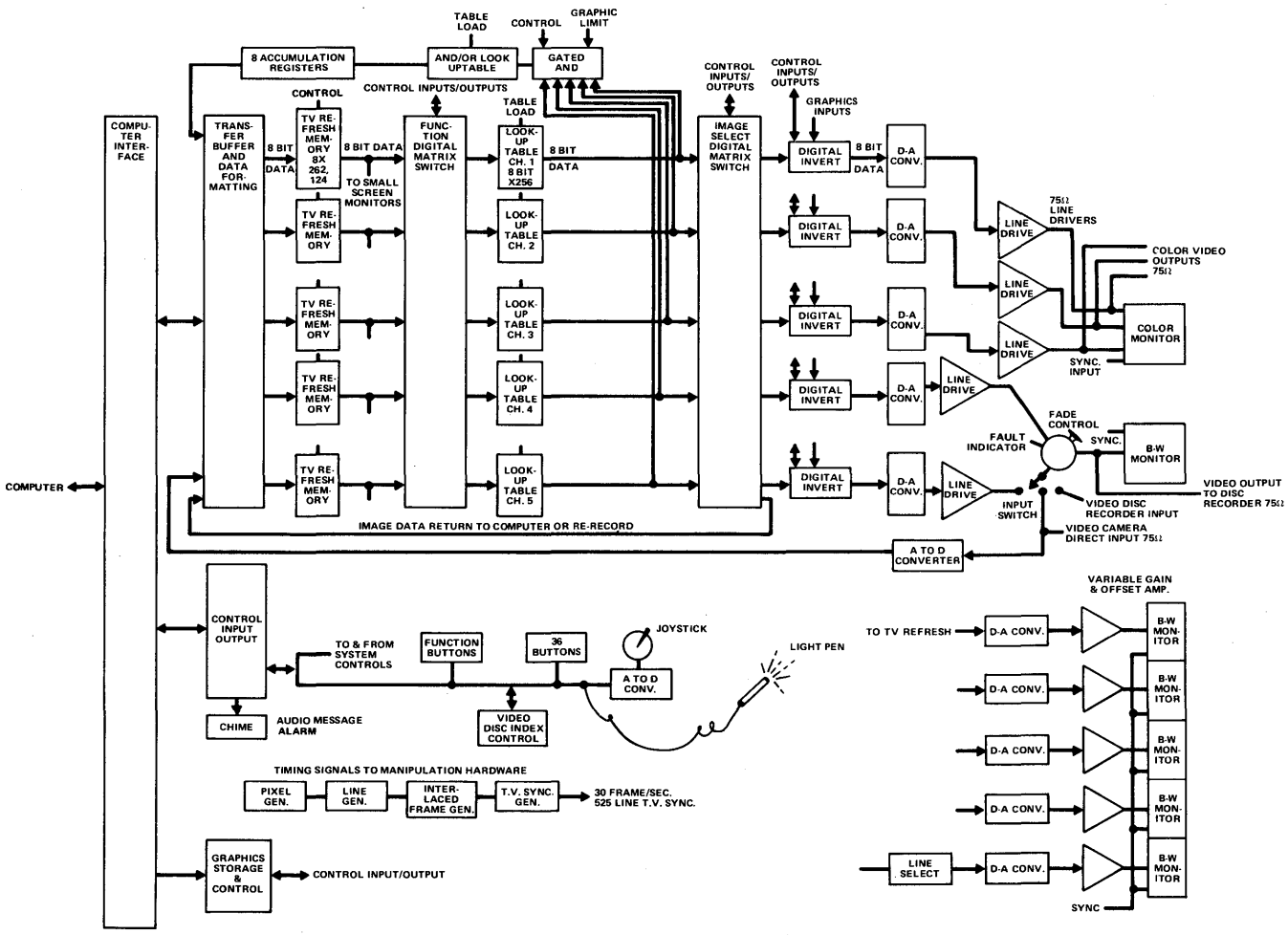


Figure 6—Terminal 2/3 logic diagram

- Scanner Input—A TV input scanner is utilized to digitize information and store it in any refresh memory channel.
- Video Transfers—All video signals can be transferred to or from the AOIPS video switching unit and the video disk unit.
- Direct Data Readout—Manipulated images and picture elements within a boundary may be transferred back to the PDP-11/70 computer for further analysis.
- Processing Control—A CRT keyboard display unit and an array of programmable function buttons are used to control the sequence of processing operations and to create new analysis scenarios.

The terminal hardware performs computer controlled image manipulation functions in real time while the user observes each operation. Image transfers from the computer to the terminal refresh memories occur at a rate of two images per second from disk units and at five images per second from the AOIPS High Density Digital Tape unit. Terminals 2 and 3 are fully interconnected to share resources thus allowing a user on either terminal to use up to 10 refresh memories to support data analysis operations.

Terminal 2 is used primarily for meteorological applications in support of severe storms research investigations. Terminal 3 is dedicated to information extraction technique development in support of severe storm forecasting investigations utilizing data from multiple satellite sensors and related surface truth measurements.

High Density Digital Tape (HDDT) subsystem

The HDDT unit is interfaced to the AOIPS PDP-11/70 and consists of a Honeywell Model 96 instrumentation tape recorder, a Martin-Marietta Data Converter, a Serial Controller Interface (SCI) fabricated by General Electric Company, and a DEC Model DR-70 Massbuss Adapter to operate the recorder on the AOIPS system.

The recorder uses one-inch wide magnetic tapes to record data on 10 of 14 data tracks. The remaining four tracks are

utilized for time tagging, audio, and two spares. The unit operates with data transfer rates of up to 20 megabits per second with bit error rates of 1 in 10^6, and provides for storage of approximately 1.4×10^{10} bits per 7200-foot reel of tape at a packing density of 20,250 bits per inch per data track.

Six playback speeds are provided, ranging from 120 inches per second to 3¾ inches per second. The unit will be operated at 7½ inches per second for playback of data into the AOIPS yielding an input data rate of approximately 1.5 megabits per second. A high-speed search capability is available at 120 inches per second; tape rewind occurs at 200 inches per second.

The Serial Controller Interface (SCI) provides the electronics needed to synchronize the serial input-output bit stream. The AOIPS SCI has been modified to subsample or average image data to allow selection of a portion of an image during data transfers to the AOIPS. This SCI capability provides options for obtaining subimages and reduced images from the full image data sets stored on high-density tapes.

SYSTEM SOFTWARE

Numerous software packages are available on AOIPS; some are geared to the specific requirements of an application and some are of a general support nature. Figures 7 and 8 present an overview of the major software packages available.

Both the PDP-11/45 (Terminal 1) and the PDP-11/70 (Terminals 2 and 3) operate under the DEC-supplied RSX-11D multiprogramming operating system. All AOIPS applications packages interface with the user through the previously mentioned hierarchical structure of menus.

Terminal 1 (the Image 100) was delivered with an application package (the Image 100 software system) designed

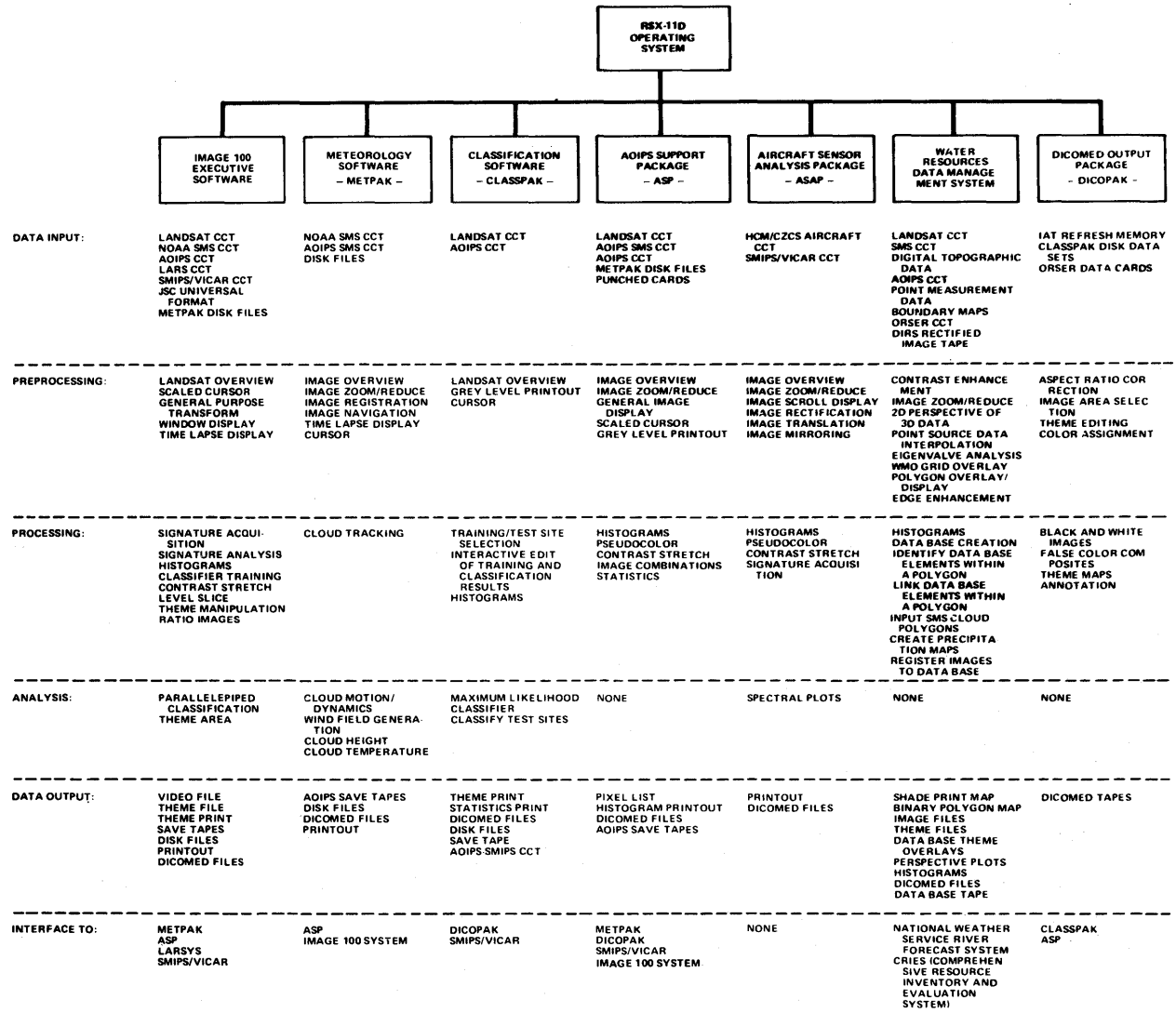


Figure 7—AOIPS Terminal 1 software overview

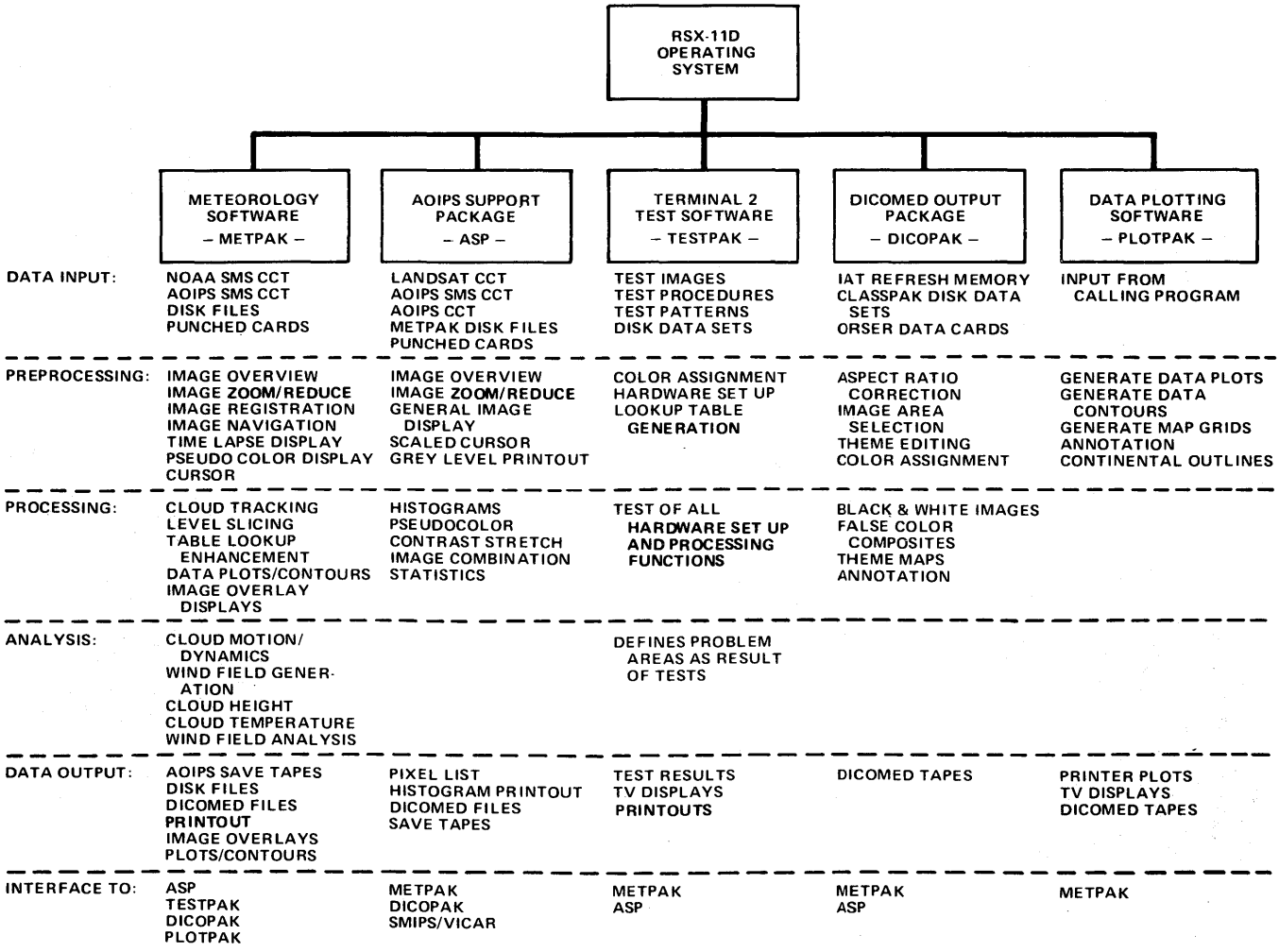


Figure 8—AOIPS Terminal 2/3 software overview

for interactive multispectral classification of Landsat satellite data. To support the varied requirements of system users, several additional AOIPS software packages were developed including:

METPAK—An interactive system which provides image navigation, registration and other functions required to support cloud motion studies, wind vector generation, and wind vector analyses. METPAK computes the cloud height and the displacement of selected clouds in a series of time lapsed digital images. It generates various analyses of the computed wind fields and displays TV and hard copy output products of images overlayed with plots and data contours of analysis parameters.

CLASSPAK—An interactive maximum-likelihood multispectral classification package. CLASSPAK accepts up to 24 channels of input data; provides for statistical analysis of the input data; and performs a classification using up to 8 selected channels. CLASSPAK provides a highly flexible, interactive environment for selecting, defining, displaying, editing, and combining training sites and training site statistics. CLASSPAK generates classification maps of up to 25 classes and displays of class correlation and covariance matrices as well as statistical analysis results.

Aircraft Sensor Analysis Package (ASAP)—An interactive software package for displaying, processing, correcting and enhancing multispectral aircraft scanner data. ASAP produces hard copy image output products and TV displays of processed aircraft imagery.

AOIPS Support Package (ASP)—An interactive system which provides general image display and manipulation capabilities. ASP performs histogram generation, contrast stretching, pseudo color enhancement, and image combination operations.

Figures 7 and 8 also list the capabilities of a number of other software packages used to manage AOIPS data bases and to generate final output products of image analysis results.

SYSTEM APPLICATIONS

During the past year, over 50 applications investigations have been supported on AOIPS. These investigations in-

Figure 9—Satellite interpretation of the dynamics of severe local storms

cluded research studies in meteorology, oceanography, hydrology, land use, forestry, geology, agriculture and related earth resources disciplines.

Reference 3 details the AOIPS capabilities used in support of severe storms investigations, particularly in the area of wind field generation and analysis. Additional system capabilities have been developed to analyze the dynamics of rapidly forming storm cells, to generate precipitation estimates from digital imagery, and to provide temperature profiles of selected clouds. These capabilities were used to produce the applications results illustrated and described in Figures 9 and 10.

The two primary earth resources applications supported on AOIPS include image enhancement for identification of ground features of interest and multispectral classification. Figure 11 describes one hydrological land use classification study conducted on AOIPS.

FUTURE SYSTEM ENHANCEMENTS

In early 1978, the AOIPS PDP-11/70 will be interfaced through a shared RP06 disk unit to another PDP-11/70 processor as shown in Figure 12. This second processor will be dedicated to deriving atmospheric temperature and humidity profiles from GOES-D VISSR Atmospheric Sounder (VAS) satellite data as part of the VAS Demonstration project. The shared disk link between the AOIPS and VAS 11/70s will provide the means of incorporating sounding data into severe storms analyses conducted on AOIPS.

The figure also illustrates plans for acquiring and processing SMS/GOES satellite data in near real time. These capabilities will allow investigators to detect and analyze severe storms as the phenomena occur in order to develop improved storm detection and prediction capabilities for the future.

During the next two years, several other AOIPS enhancements are planned for meteorological and earth resources applications including:

- Enhancing capabilities for monitoring the dynamics of cloud formation and severe storm development.
- Improving precipitation estimation capabilities using several estimation techniques.
- Developing capabilities for multispectral analysis of severe storm data.
- Applying multispectral classification techniques to severe storm analyses.

FIGURE 1: 21:08 GMT
SMS-2 OMAHA THUNDERSTORM

ENLARGED AREA HIGHLIGHTS THE CLOUD TOP TEMPERATURE STRUCTURE OF THE THUNDERSTORM CELL AT THIS INSTANT IN TIME.

FIGURE 2: 21:28 GMT
SMS-2 OMAHA THUNDERSTORM

ENLARGED AREA HIGHLIGHTS THE CLOUD TOP TEMPERATURE STRUCTURE OF THE THUNDERSTORM CELL. BOTH THE 206°K AND THE 208°K AREAS HAVE INCREASED RAPIDLY. THE OMAHA TORNADO OCCURS AT 21:33 GMT.

INTRODUCTION

THE USE OF GEOSTATIONARY SATELLITE DATA HAS THE POTENTIAL OF BEING AN IMPORTANT COMPONENT IN THE DETECTION AND MONITORING OF THUNDERSTORMS AND OTHER FORMS OF SEVERE WEATHER. THE OBJECTIVE OF THIS RESEARCH IS TO DETERMINE IF THUNDERSTORM GROWTH RATES AND CHARACTERISTICS OBSERVABLE WITH SYNCHRONOUS METEOROLOGICAL SATELLITE (SMS) INFRARED DATA CAN BE LINKED WITH THE OCCURRENCE OF SEVERE WEATHER ON THE GROUND (TORNADOES, HAIL, HIGH WINDS). THESE SEVERE WEATHER EVENTS ARE HIGHLY CORRELATED WITH THE INTENSITY OF ATMOSPHERIC CONVECTION. BY ANALYZING SMS INFRARED DATA ON THE ATMOSPHERIC AND OCEANOGRAPHIC INFORMATION PROCESSING SYSTEM (AOIPS), PARAMETERS RELATED TO CONVECTION INTENSITY SUCH AS CLOUD TOP TEMPERATURE AND HEIGHT AND THE RATE OF VERTICAL GROWTH OF A THUNDERSTORM CAN BE IDENTIFIED.

AOIPS ANALYSIS

THE ATMOSPHERIC AND OCEANOGRAPHIC INFORMATION PROCESSING SYSTEM (AOIPS) WAS APPLIED TO ANALYZE SMS DATA TAKEN AT FIVE MINUTE INTERVALS BETWEEN 1803 AND 2208 GMT ON MAY 6, 1975 OVER NEBRASKA. A NUMBER OF SEVERE STORMS THAT SPAWNED TORNADOES AND HAIL DURING THIS TIME PERIOD CAN BE SEEN ON THIS SMS DATA. FIGURES 1 AND 2 ARE TWO IMAGES (20 MINUTES APART) EXTRACTED FROM THE TIME SERIES. THE THUNDERSTORM CELL THAT SPAWNED A TORNADO OVER OMAHA IS OUTLINED BY THE PURPLE BOX IN BOTH FIGURES WHERE IT IS ALSO ENLARGED 16 TIMES AND DISPLAYED IN COLOR TO HIGHLIGHT THE DYNAMICS OF THE CLOUD TOP TEMPERATURE STRUCTURE. THE GREEN AND BLUE REGIONS (208°K AND 206°K) INDICATE THE COLDEST OR HIGHEST PORTION OF THE THUNDERSTORM. THE INCREASE IN AREA OF THESE COLD REGIONS FROM FIGURE 1 TO FIGURE 2 IMPLIES STRONG CONVECTION AND A RAPID INCREASE IN THE HEIGHT OF THE THUNDERSTORM TOP.

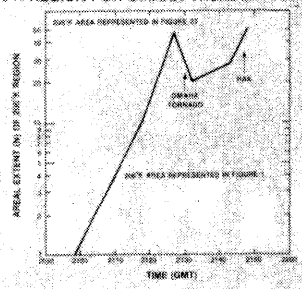
THE GROWTH OBSERVED IN FIGURES 1 AND 2 IS PRESENTED QUANTITATIVELY IN FIGURE 3 AS THE TIME INCREASE IN AERIAL EXTENT (N) OF THE 206°K REGION OF THE STORM CELL. THIS 206°K REGION INCREASES RAPIDLY JUST PRIOR TO THE OCCURRENCE OF THE OMAHA TORNADO AND AGAIN PRIOR TO OBSERVED HAIL. MOST OF THE OTHER SEVERE STORMS ANALYZED IN THIS STUDY EXHIBITED SIMILAR GROWTH PATTERNS IN THE CLOUD TOP TEMPERATURE STRUCTURE. THESE TRENDS ARE READILY RECOGNIZED WHEN PLOTTED AS AN INDEX OF SEVERE WEATHER AS IN FIGURE 4. IN THIS FIGURE, THE NORMALIZED RATE OF COLD AREA GROWTH $\frac{\Delta N}{N \Delta t}$ IS PLOTTED AGAINST CLOUD TOP TEMPERATURE FOR EACH OF THE 39 SEVERE AND NON-SEVERE CASES STUDIED.

A LARGE MAJORITY OF THE CASES STUDIED THAT FALL TO THE RIGHT AND ABOVE THE DIAGONAL DISCRIMINATION LINE ARE ASSOCIATED WITH SEVERE WEATHER WHILE THE CASES FALLING ON THE OTHER SIDE OF THE LINE ARE PREDOMINANTLY NON-SEVERE.

CONCLUSIONS

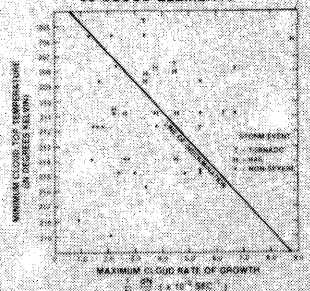
DIGITAL INFRARED DATA FROM A GEOSTATIONARY SATELLITE (SMS-2) WERE ANALYZED ON AOIPS TO STUDY THUNDERSTORM DYNAMICS IN RELATION TO THE OCCURRENCE OF SEVERE WEATHER ON THE GROUND. AN EXAMINATION OF 39 CLOUD ELEMENTS INDICATED THAT PARAMETERS DERIVABLE FROM SMS SHORT INTERVAL DATA ARE CORRELATED WITH THE OCCURRENCE OF SEVERE WEATHER. THESE PARAMETERS INCLUDE THE CLOUD TOP MINIMUM TEMPERATURE AND RATE OF VERTICAL GROWTH. THEREFORE, IT APPEARS THAT SHORT INTERVAL INFRARED DATA FROM SMS MAY BE USEFUL IN CALCULATING PARAMETERS RELATED TO THUNDERSTORM INTENSITY AND THE OCCURRENCE OF SEVERE WEATHER ON THE GROUND.

FIGURE 3: PLOT OF AREAL EXTENT OF THE 206°K REGION FOR OMAHA THUNDERSTORM



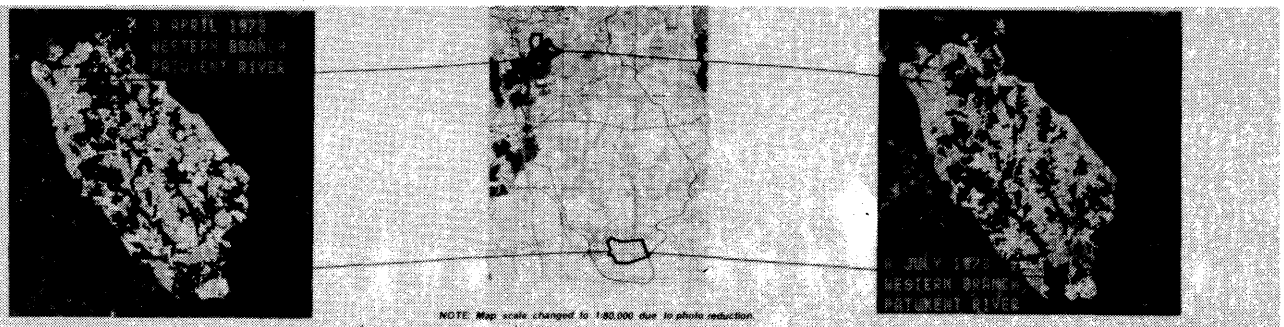
RAPID GROWTH OCCURS PRIOR TO BOTH THE OUTBREAK OF TORNADO AND HAIL EVENTS

FIGURE 4: PREDICTION INDEX PLOTTED FROM STUDIES OF 39 CLOUD ELEMENTS



A MAJORITY OF THE SEVERE WEATHER OCCURRENCES FALL TO THE RIGHT AND ABOVE THE DIAGONAL LINE OF DISCRIMINATION

Figure 10—Thunderstorm intensity parameters viewed from satellite



Imagery from Landsat-1 has been analyzed for hydrologic land use classification using the GSFC Atmospheric and Oceanographic Information Processing System (AOIPS). The upper portion of the Western Branch subbasin of the Patuxent River watershed served as a test site for classification. The boundary of this subbasin is identified on the 7½-minute USGS topographic map. The classification of Landsat image 1260-15201, 9 April 1973 and that of Landsat image 1350-15192, 8 July 1973 each resulted in cover type identification of greater than 90 percent of the watershed surface. These classifications (relative to the watershed boundary) are being used by the Hydrology and Oceanography Branch as input to hydrologic models for stormwater runoff estimation.

The AOIPS is a user-interactive computer system designed for analysis of remotely sensed earth resources information. Capabilities include various types of classification packages applicable to interpretation of data from ATS, Nimbus, GOES, ESMR, HCMM, side-looking radar, aircraft multispectral scanners and digitized photography as well as Landsat. The system rapidly analyzes Landsat and other imagery with results of each 'training', or data sample classification, being displayed within seconds on the cathode ray tube for user evaluation and modification. Each of the final watershed classifications shown here was completed within twenty hours of system operation.

CLASSIFICATION RESULTS

CLASS	APRIL		JULY	
	Hectares	Percentage	Hectares	Percentage
Agric fields	1116.8	15.5	2421.4	33.6
Construction	a		153.2	2.1
Water	23.0	0.32	18.7	0.26
Woodland	3194.9	44.5	2989.7	41.6
Residential	864.1	12.0	809.5	11.3
Urban	18.1	0.25	35.5	0.49
Bare fields	1297.2	18.1	261.4	3.6
Total classified	6495.6	90.7	6709.4	93.3
Watershed	7168.0		7168.0	

a No classification for construction sites was attempted on the April image.

The table at the left contains area measurements for each of the landcover classes identified within the Western Branch subbasin. Also given is the percentage of the subbasin covered by each class. Note that the major difference between the two classifications is due to seasonal vegetation coverage changes. April's 'bare fields' are classified as 'agricultural fields' in the July image. The growth in riparian vegetation has caused a slight decrease in the surface of water exposed. Except for the fact that some of the urban surfaces were left unclassified in the April scene, there is a high degree of precision between the AOIPS classifications from two different images.

Figure 11—Hydrologic land use classification of Landsat imagery

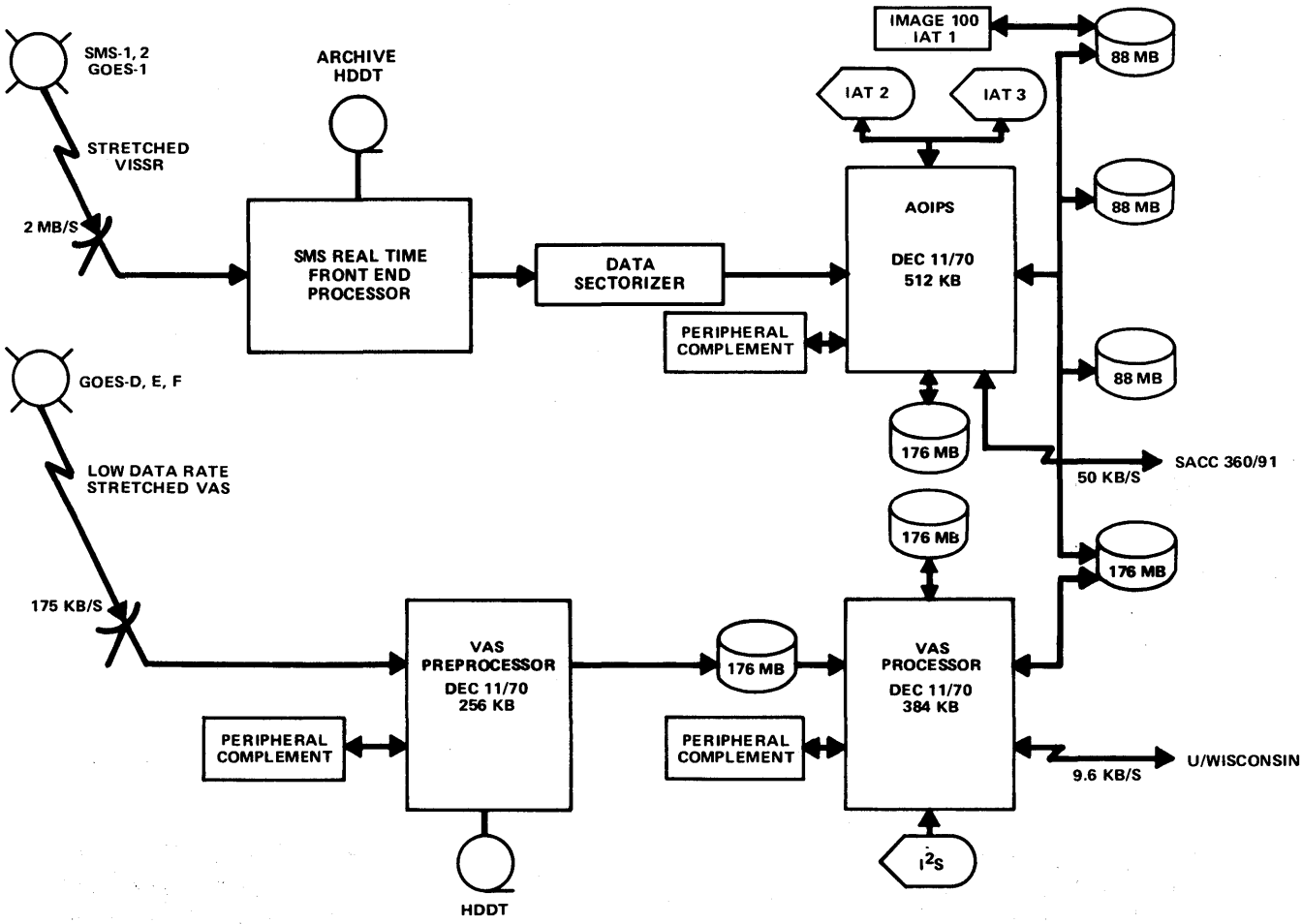


Figure 12—AOIPS—VAS system configuration

- Incorporating a wider range of nonimage surface truth and ancillary information into AOIPS information extraction operations.
- Providing capabilities for registering image and nonimage data from multiple sources.
- Establishing integrated data bases for storage, comparison and correlation of multisource/multitime image and nonimage data.

An array computer called the Massively Parallel Processor (MPP) will also be interfaced to AOIPS in 1981. When completed, the MPP will utilize an array of 128×128 primitive processing elements (i.e., elementary central processing units) to perform image processing computations on 16,384 picture elements simultaneously. The MPP has been designed to operate at speeds of over 5,000 million additions per second and over 1,000 million multiplications per second on eight bit integers. The system will be fully programmable and capable of handling variable word lengths and floating point computations.

An 8×8 breadboard version of the MPP has been successfully tested and is scheduled to be interfaced to AOIPS in April 1978. Reference 4 provides detailed information on the design and functional characteristics of the MPP.

SUMMARY

This paper has discussed the configuration and processing capabilities of the AOIPS interactive image processing system. Unique subsystems for displaying, analyzing, storing and manipulating digital image data were presented. Applications of the AOIPS to research investigations in meteorology and earth resources were highlighted.

Our experience to date indicates that future progress in relating remotely sensed measurements to earth based phenomenon is heavily dependent on an investigators ability to interact with his data and to combine multisensor/multitemporal data sets in integrated analysis data bases. In most cases, this interaction must take place within hours or days of the occurrence of an event to preserve the value of the information content in the data and to maximize the realizable benefits.

REFERENCES

1. Dalton, J. T., "Configuration Description and Load Analysis for the Atmospheric and Oceanographic Information Processing System (AOIPS)," NASA/GSFC Report X-933-75-217, July 1975.
2. Bracken, P. A., J. T. Dalton, J. B. Billingsley and J. J. Quann, "Atmospheric and Oceanographic Information Processing System (AOIPS)—System Description," NASA/GSFC Report X-933-77-148, March 1977.
3. Billingsley, J. B., "Interactive Image Processing for Meteorological Applications at NASA/Goddard Space Flight Center," *Proceedings of Seventh Conference on Aerospace and Aeronautical Meteorology,* Symposium on Remote Sensing from Satellites, Melbourne, Florida, November 1976.
4. Fung, L., "A Massively Parallel Processing Computer," *Proceedings of the Symposium on High Speed Computers and Algorithm Organization,* University of Illinois, Urbana, Illinois, April 1977.

Remote sensing program in earth resources

by FREDERIC C. BILLINGSLEY
NASA Headquarters
Washington, DC

and

DONALD T. LAUER
U.S. Geological Survey
Sioux Falls, South Dakota

THE EARTH RESOURCES SITUATION

Population growth and demands for higher standards of living are creating pressures which affect all aspects of our society. The global population, currently near 3900 million, is growing at a rate of 75-80 million/year.

Coupled with population growth, and magnifying the effect, are rising demands for a more equitable sharing of the products and benefits of technology. The products and benefits derive from the consumption of natural resources. As societies develop and technological advancement begins, individual demands increase resulting in a more rapid rate of consumption of resources. Information is necessary to enable prudent decisions regarding the management of resources. But understanding alone will not lead to solutions on a long-term basis. Knowledge must be current, necessitating continuing inputs of fresh information. The most practical and economical manner of acquiring repetitive global, Earth Resources information is by means of a satellite-based remote sensing system.

REMOTE SENSING DATA SYSTEM

1. *General*

A total remote sensing data system (wherever the components are located and whatever the sophistication) comprises four distinct operations:

- Physical data acquisition and preprocessing
- Archiving for bulk retrieval
- Production and dissemination of individual products
- Extraction of the required information from the data

2. *Data Acquisition*

Mechanisms for data acquisition are the same for most applications. Global, repetitive, timely, and accurate information is a key element. Landsat addresses, and to a considerable extent, can satisfy the requirements for information. For this reason, the discussions will concentrate on Landsat data, although recognizing the presence of other sensor platforms in the NASA program. Not part of the system considered here, but often critical to successful solution of a given problem, is the inclusion of data from many other sources, such as maps or socioeconomic data. It is the task of the data analyst to blend all necessary data, from whatever source, to extract the desired information, and to present the results to the decision maker in the most optimum form.

As Landsat data are received by the data collection stations, they are forwarded to Goddard Space Flight Center (GSFC). Until July 1978, GSFC will apply radiometric and geometric corrections in making 70mm film masters, which are sent to the data archive at the Earth Resources Observation System (EROS) data center (EDC) in Sioux Falls, South Dakota. GSFC can also produce computer compatible tapes (CCT) on special order (through EDC) in which radiometric correction has been applied. Beginning in 1978, GSFC will deliver digitally corrected digital data to EDC on high density bulk tapes as the standard NASA product.

3. *Archiving*

All data are archived at EDC, where they are available for reproduction upon request of orders from users. A customer service and catalog function is available at EDC to assist users in locating required data.

4. *Availability of Landsat data*

Landsat data are currently reproduced and distributed to users by three Federal Data Centers operated separately by the U.S. Department of Interior's Geological Survey (USGS), the U.S. Department of Agriculture (USDA) and the U.S. Department of Commerce's National Oceanic and Atmospheric Administration (NOAA). Data products provided by the three centers are similar in scale, format and type. By mid 1978 a new all-digital system for handling and processing Landsat data will be in operation and will be providing improved products to data users. A processing

system will be implemented whereby the National Aeronautics and Space Administration (NASA) will provide Landsat high-density digital tapes (HDTs) to the USGS's EROS Data Center (EDC) which will be the nation's federal dissemination facility for original Landsat data products.

The Landsat high-density digital tapes routinely received at EDC from NASA's Goddard Spacecraft Center will be radiometrically restored (for detector gain and offset) and geometrically corrected (to a Space Oblique Mercator map projection) using a cubic convolution resampling algorithm. By means of retrospective user orders, NASA will provide EDC Landsat data which are corrected to the Universal Transverse Mercator or the Polar Stereographic (over 65° latitude only) map projection, resampled with a nearest neighbor algorithm, or in an uncorrected high-density tape format (i.e., radiometrically restored but not geometrically corrected).

The digital processing system at EDC has the flexibility to produce a brightness-frequency histogram showing brightness value distribution for each Landsat scene, and perform contrast and edge enhancement functions which are based on the brightness value distributions. When placing an order, a user may select or omit these enhancement functions. Film and digital tape products, from the multispectral scanner (MSS) and the return beam vidicon (RBV) camera, will be available from EDC. Negative film working masters in a 241mm format will be made on a high resolution laser-beam film recorder. Customer products made from the film masters will include second generation positive film transparencies, third generation negative film transparencies, black-and-white or color composite prints at various scales of enlargement, and 70mm film products. Computer compatible tapes (CCTs, 800 or 1600 bits per inch) generated from high-density digital tapes will include RBV data in subscene sequential format, or MSS data in band sequential or line interleaved formats. Pixel interleaved format will no longer be available. Specified images may be selected and specially ordered in an HDT format.

More information concerning these data products may be obtained from:

User Services
EROS Data Center
U.S. Geological Survey
Sioux Falls, SD 57198

5. *Foreign Ground Stations*

In addition to the data returned to the United States and available through EDC, several countries have elected to install their own receiving stations. At the present time, these are Canada, Brazil, and Italy. In addition, agreements have been reached with Iran (station under construction), India, Argentina, Zaire, and Chile (the latter two are not yet funded).

NASA and the U.S. climate program—A problem in data management

by JOHN J. QUANN
NASA/Goddard Space Flight Center
Greenbelt, Maryland

INTRODUCTION

Climate can be thought of as the average of weather. Whereas fluctuations in weather on a daily basis are essentially unpredictable, fluctuations in weather over longer time scales are part of climate and may well be predictable. The goal of the U.S. Climate Program is to help the nation respond more effectively to climate induced problems by enabling its government to be aware of or to anticipate climatic fluctuations and their domestic and international impacts and to identify the real and potential impact of man on regional and global climate.[1]

In order to deal with the many aspects of climate, NASA developed an agency plan, which defines the NASA contribution to the U.S. Climate Plan. In it, NASA chose to divide climate into four separate but related activities:[2]

1. Climate A deals with the current state of the climate. The goal is to provide timely information in a usable form on the current state of the climate and to provide for the analysis of this information.
2. Climate B deals with regional climate on time scales longer than a month but shorter than a decade. The goal is to predict fluctuations in climate over individual regions of several hundred kilometers square.
3. Climate C refers to climate which occurs on time scales of a decade and beyond. The goal is to understand the causes of natural changes in the global climate.
4. Climate X deals with the climate as it is affected by mans activities on all temporal and spatial scales. The goal is to understand and anticipate the impact of man so that appropriate action can be taken.

Currently, programs related to the analysis and modelling of climate are limited and dispersed throughout many agencies in the government. The National Oceanic & Atmospheric Administration (NOAA), The National Science Foundation (NSF), The Department of Defense (DOD), and the National Aeronautics & Space Administration (NASA) are but some of the agencies and institutions in which research into climate is being conducted. However, there is no central focus at the present time. No group is responsible for establishing and maintaining a comprehensive program dealing with the many aspects of climate. This situation will change with the establishment and implementation of a national climate plan which will be integrated not only among domestic agencies but among many nations having related programs. At present the major agencies involved are developing, in parallel, the plans for their respective agencies to participate in the overall program. NASA's role in the achievement of the objectives of the National Climate Plan relates primarily to satellite based instrumentation, the development of climate required parameters on a global scale, climate modelling and assessment, special studies, and data management.

DATA MANAGEMENT

The objective of the climate data management system is to permit effective use of a global data base for climate research. Information contained in the data base must allow researchers to answer a variety of questions on climate . . . questions related to the impact of climate variations on crop yields, energy, land use and water resources; questions relating to the determination and effect of gas and particle loading in the atmosphere; questions concerning the variability of climate and the relationship to the earth's energy budget, distribution of cloudiness, snow cover etc.

The global data base must facilitate research on a wide range of time and space scales. It must be able to aid in the development of climate models and provide data for their verification. It must support diagnostic studies that investigate the structure of climate changes and facilitate assessment studies as the effect of climate change on food and fiber production.[3]

NASA'S ROLE

Forty (40) parameters have been identified as key variables requiring observation from space and having a direct effect on climate (Table I). NASA's contribution to the total climate data base will be to produce climate data sets from its experimental space observing systems and to maximize the value of this data for climate analysis and prediction. Validated data sets will be provided to NOAA for inclusion

TABLE I

NO.	Parameter
Weather Variables	
1.	Temperature Profile
2.	Surface Pressure
3.	Wind Velocity
4.	Sea Surface Temperature
5.	Humidity
6.	Precipitation
7.	Cloud Cover
8.	Boundary Layer Stability
Ocean Parameters	
4a	Sea Surface Temperature
9.	Evaporation
10.	Surface Sensible Heat Flux
11.	Wind Stress
12.	Sea Surface Elevation
13.	Upper Ocean Heat Storage
14.	Temperature Profile
15.	Velocity Profile
Radiation Budget	
7a	Clouds (Effect on Radiation)
16.	Regional Net Radiation Components
17.	Equator-Pole Gradient
18.	Surface Albedo
19.	Surface Radiation Budget
20.	Solar Constant
21.	Solar Ultraviolet Flux
Land, Hydrology, and Vegetation	
6a	Precipitation
18a	Surface Albedo
22.	Surface Soil Moisture
23.	Soil Moisture (Root Zone)
24.	Vegetation Cover (Non-Forest)
25.	Evapotranspiration
26.	Plant Water Stress
Cryosphere Parameters	
27.	Sea Ice (Percent Open Water)
28.	Snow (Percent Coverage)
29.	Snow (Water Content)
30.	Ice Sheet Surface Elevation
31.	Ice Sheet Horizontal Velocity
32.	Ice Sheet Boundary
Atmospheric Composition	
21a	Solar Ultraviolet Flux
33.	Stratospheric Aerosol Optical Depth
34.	Tropospheric Aerosol Optical Depth
35.	Ozone
36.	Stratospheric H_2O
37.	N_2O,NO_x
38.	CO_2
39.	CFM's
40.	CH_4

in their overall diagnostic data base. NOAA will, in turn, provide NASA similar data sets from both conventional data sources and operational space observing systems. Standard weather parameters such as surface temperature, surface pressure, temperature profiles, wind velocity and humidity will be acquired through the operational observations of the World Weather Watch (WWW) Program and will be stored and accessed through NOAA's Environmental Data Services (EDS). NASA and NOAA space observation data has been reviewed and a decision made that the starting point for a climate data base would be January 1973 which coincides with the launches and data availability from the NASA Nimbus 5 and the NOAA-1 satellites. The sources of data from existing and planned satellite programs required for Climate B are summarized in Table II. Additional sources for Climate C and X are given in Table III. It is unlikely that full extraction of climate parameters can be afforded from all sources. As an example, the Scanning Radiometer on the NOAA satellites is the most consistent source for global cloud data at the present time. Cloud information could also be obtained from the Nimbus and Landsat satellites but the data coverage is insufficient. Another potential source for a cloud data set is the geosynchronous SMS (Synchronous Meteorological Satellite) and GOES (Geostationary Operational Environmental Satellites). When all geosynchronous spacecraft are in place only the polar regions will not be covered. It may be feasible at that time to extract only the polar regions from polar orbiting spacecraft.

DATA BASE

The information contained in the data base will be accessible at four different levels of processing. Level O is raw data as telemetered from the spacecraft. Level I consists of data which has been calibrated into engineering units, e.g., radiances, microwave brightness temperatures, and located with respect to time, orbit, and attitude. Level II refers to climate parameters, e.g., sea surface temperature, percent sea ice, soil moisture, at full spatial and temporal resolution.

Acronyms

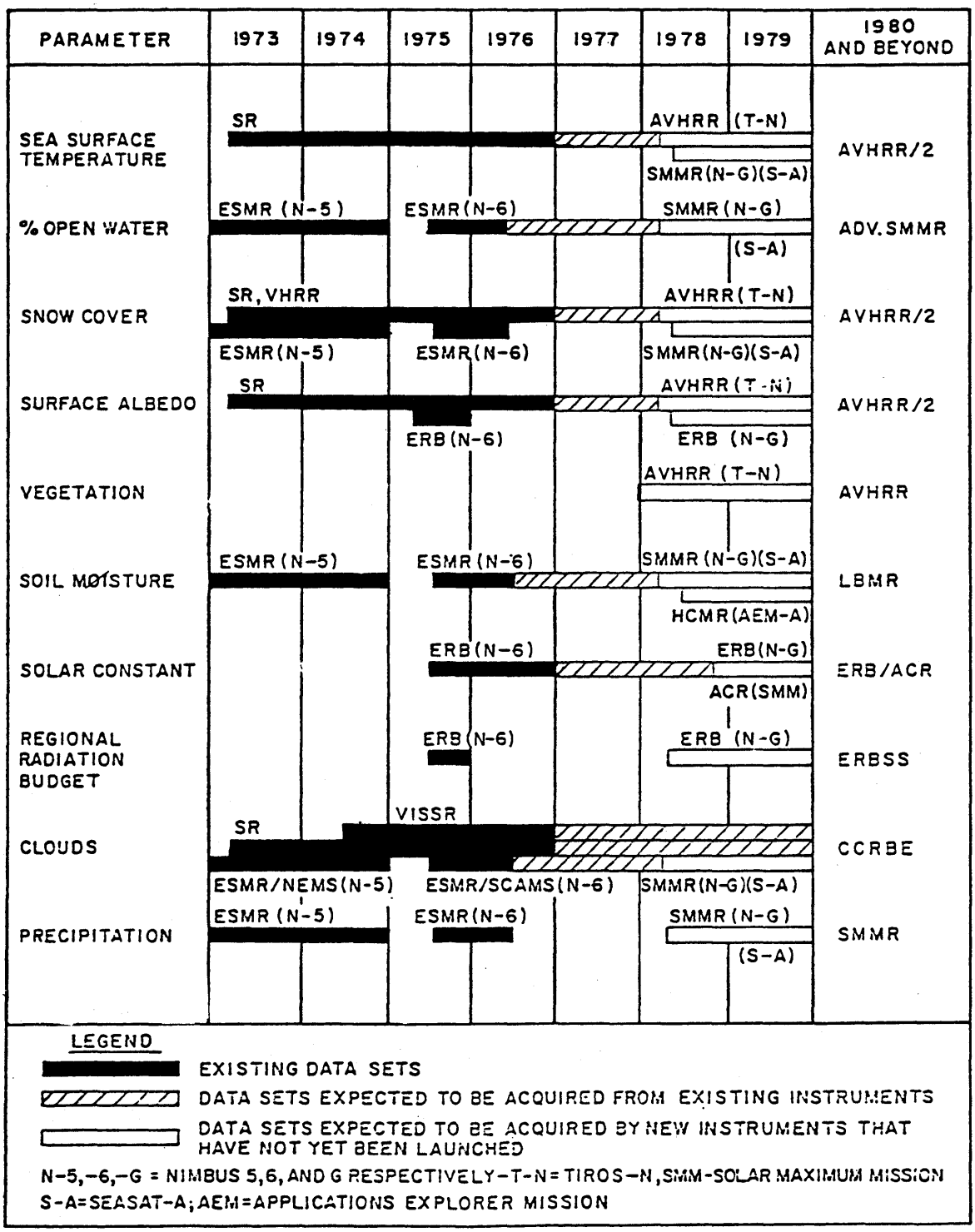
1. ACR	—Active Cavity Radiometer
2. AVHRR	—Advanced Very High Resolution Radiometer
3. CCRBE	—Cloud Cover Radiation Budget Experiment
4. ESMR	—Electrically Scanning Microwave Radiometer
5. ERB	—Earth Radiation Budget
6. HCMR	—Heat Capacity Mapping Radiometer
7. LBMR	—L-Band Soil Moisture Radiometer
8. LIMS	—Limb Infrared Monitor of the Stratosphere
9. LRIR	—Limb Radiance Inversion Radiometer
10. SAGE	—Stratospheric Aerosol & Gas Experiment
11. SAMS	—Stratospheric & Mesospheric Sounder
12. SAMII	—Stratospheric Aerosol Measurement
13. SBUV	—Solar Backscatter Ultraviolet instrument
14. SMMR	—Scanning Microwave Multichannel Radiometer
15. SR	—Scanning Radiometer
16. VISSR	—Visible & Infrared Spin Scan Radiometer

Level III consists of climate parameters which have been spatially and temporally averaged.

Functionally, the climate data management system must:

1. Acquire raw telemetry data from the applicable NASA sensors.
2. Catalog, store, and retrieve all information archived in the NASA data base at any level.
3. Calibrate, locate, and extract climate related parameters from the raw data.
4. Facilitate the validation and analysis of information.
5. Manage and give easy user access to a complete, current NASA data directory.
6. Grid, register, spatially and temporally average climate parameters.

TABLE II—Basic Data Sources for Climate B

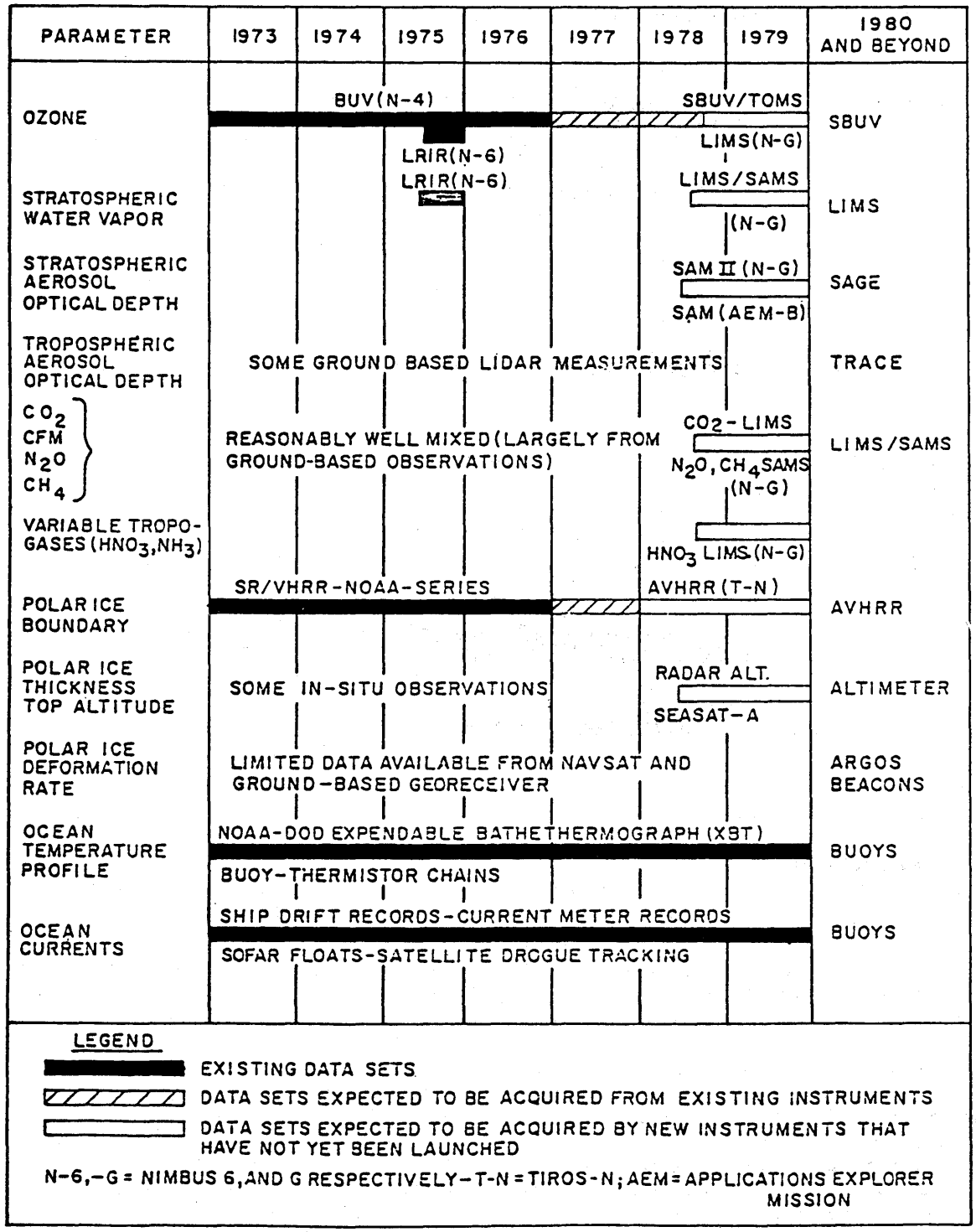


7. Provide information to users in pre-selected formats or as experimental data sets in non-standard form.
8. Provide a central interface with NOAA for access to and exchange of climate data sets outside of NASA.
9. Provide a centralized access to climate data sets in a manner which is functionally transparent to the user.
10. Provide acceptable response time to the user.

Figure 1 presents a functional overview of the NASA climate data management system. NASA Goddard will be responsible for receiving all raw spacecraft data from NASA missions and maintaining a central end to end data directory. The functions associated with the transformation of the raw data into the Level I data base is the responsibility of the NASA center managing the particular payload. As an ex-

ample the Earth Radiation Budget (ERB) Satellite is managed by Langley Research Center. Langley, therefore, would be responsible for the Level I processing and archival functions. The processing of information into individual climate parameters, i.e., Level II, is the responsibility of the NASA Center charged with development of the particular climate parameter. As an example, although Langley may have the responsibility of processing the ERB data into the Level I data base, another NASA Center, Goddard for example, may have the responsibility of processing the Level I data into a measurement of the solar constant which is a Level II climate parameter. Level II data, therefore, will be a distributed data base. Each Center has the option to maintain its own local data base for the Level II climate param-

TABLE III—Basic Additional Data Sources for Climate C&X



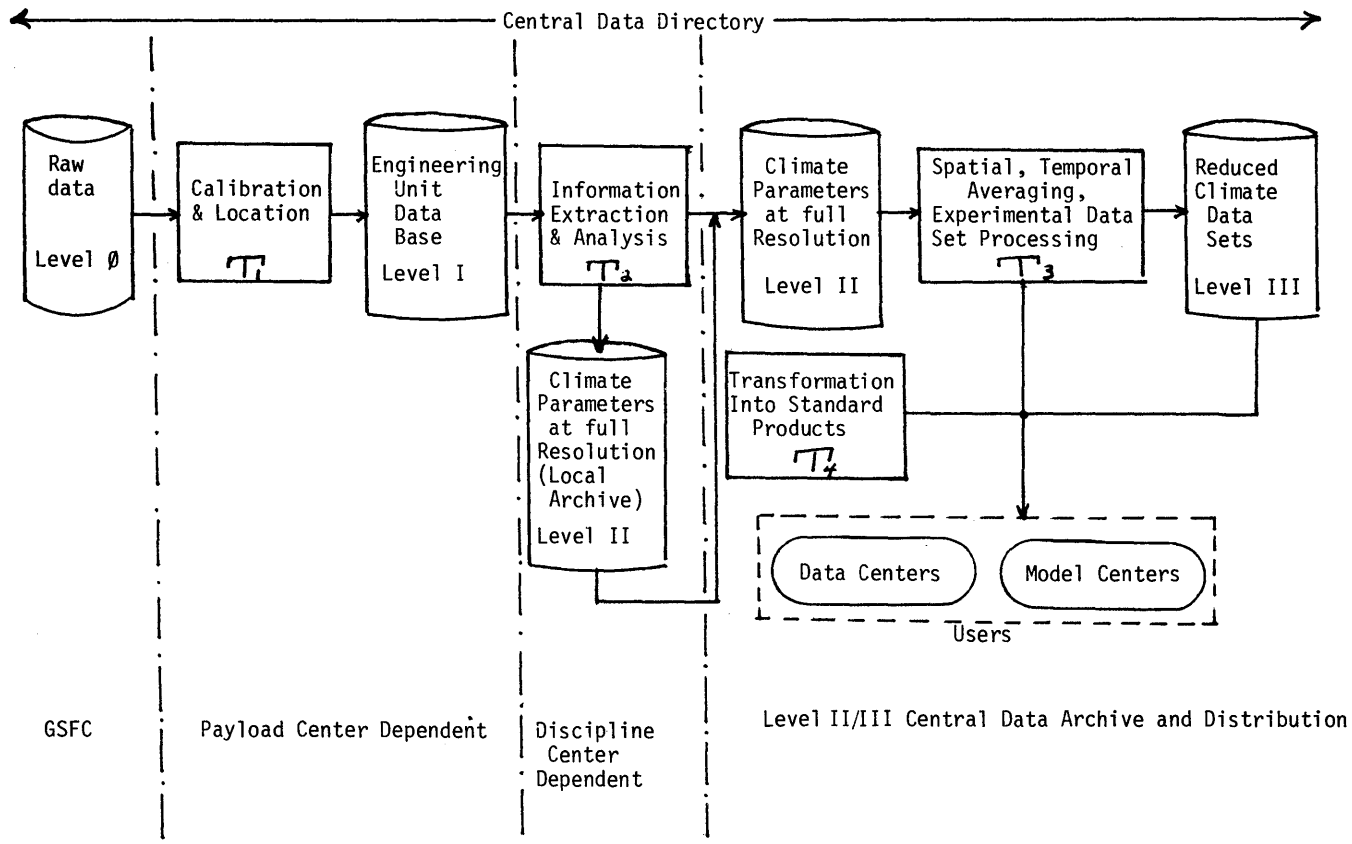


Figure 1—NASA climate research program data management

eter for which it has the responsibility of developing, or submit the data to the central data base maintained at Goddard. If the former course is chosen, the local data base must be accessible by the Goddard data management system. Regardless of where the Level II data is archived, all information concerning its availability will be maintained in the central data directory at Goddard. Level III data will be centrally archived and distributed by Goddard. In addition to maintaining the NASA data base the data management system must be able to interrogate, access, and retrieve information from the NOAA data base in order to integrate data sets in terms of temporal and spatial resolution.

SUMMARY

The Climate Program is not short term, but is anticipated to extend over decades. To catalog and manage global data in all its forms, to facilitate retrieval of large integrated and distributed data sets upon user demand, simply to store, and access the equivalent of 10^5 digital data tapes, are typical problems illustrative of the scope and magnitude of Climate data management. It will be the largest, most complex data management system ever developed by NASA. But the analysis of climate is complex and requires such a system if it is going to be effective, and effective it must be if man is to benefit.

REFERENCES

1. U.S. Domestic Council 1974: A United States Climate Program. Report of the Subcommittee on Climate Change of the Environmental Resources Committee, Washington, D.C.
2. "Proposed NASA Contribution to the Climate Program," July 1977.
3. Jenne, Roy L. "The Global Data Base for Climatic Research," Preliminary Report to the JOC, September 27, 1977.

What good is all this data if we can't use it?

by GEORGE J. McMURTRY
The Pennsylvania State University
University Park, Pennsylvania

One of the most disappointing aspects of the Landsat program to date has been the relatively slow acceptance and use of this remote sensing data by federal, state, and local agencies. In order to attract a greater number of users of remotely sensed multispectral data, several conditions must be satisfied. The user must be convinced that he has the resources and capability of acquiring, processing, and interpreting the data and that information extracted from that data will be helpful to him in satisfying the requirements of his particular job. The final processed results must provide the desired information in an understandable and attractive format. Furthermore, the user must be able to obtain the final result more cheaply, quickly, and/or easily than by conventional methods. Finally, he must be able to integrate the data and/or the processed results with other pertinent data.

In the session on Image Processing System Design, two interactive systems have been described. The word "interactive" itself implies an intimate involvement of the user. This type of analysis tends to require a more technically sophisticated user. This in turn implies that user agencies will require highly skilled and trained personnel who can interact with the appropriate processing system. Even so, there will still be a need to have interactive systems with which a user may interact "in English" using menus, etc., i.e., an input-output format which makes the system appear to be not only accessible but attractive and easy to use. If these conditions are satisfied the user must then decide which specific set of hardware and software he should purchase for installation in his own facility. Another alternative is for the user to purchase time and perhaps services at some other facility.

In addition to the papers on interactive analysis systems, we have heard a description of the design of a large pipeline system for handling large quantities of data routinely and quickly. The emphasis in such a system is to process a large volume of image data with a high through-put for distribution to a wide community of users. Such a system has additional requirements for reliability, special processing, and accuracy. It must take into account the needs and desires of a wide variety of users. In determining the types of processing available in such systems several questions are pertinent. Is it cost effective for NASA to provide the special processing required by the user? Or would it be better for the user to obtain raw data and to process that data according to his own needs? To answer these questions one must determine whether it is reasonable to expect the user to have the capability to perform such processing. Many users will not. This may suggest that if the principal objective is to increase the number of users, then any pipeline system should be designed for the needs of many relatively small users who would not be expected to have sophisticated processing capability rather than to design the system for the user with the more highly sophisticated equipment and software, even though the latter user may be the recipient of a much larger volume of data. In general, the user of whom we are speaking will need a product that has been both geometrically and radiometrically corrected and referenced to a common coordinate system. As the user community grows, the users' needs and capabilities will change and any pipeline system must be flexible and be adaptable to such changes.

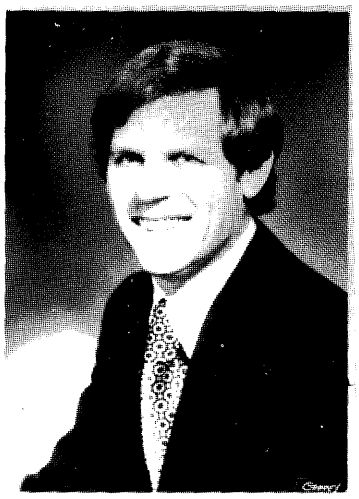
The authors of the two papers on interactive systems have stated that there is a need "to combine multisensor/multitemporal data sets in integrated analysis data bases" and for "registration of the existing geocoded data bases with Landsat imagery . . . if the Landsat data is to be truly useful to the user community." This implies that the output of a pipeline system should consist of corrected data, fully processed to a standard geodetically defined grid and it has been suggested that this data should be specifically designed to overlay groups of 7.5 minute topographical quadrangle sheets (1:24,000). Many different projections have been recommended including SOM, Lambert Conformal Conic, Polyconic, state plane coordinates, equal area, and lattitude/longitude. Taking into account user needs, conformality, zone boundary problems, and ease of mechanization, a conformal projection such as UTM should be used. The header format on data provided to users should include the following kinds of information:

1. Geometric
2. Radiometric
3. Historical
4. Statistical, including reliability and accuracy estimates.

The format should provide a structure which will handle ancillary information in the form of either cellular data, numerical lists, or alphameric lists. Interleaving of bands by

pixels or by lines should be determined by the needs of the users with less sophisticated facilities. If the data demand is sufficient, both types of interleaving could be available as standard products. Most non-research users will probably not be concerned about the types of sampling techniques used in correcting the data but they should be informed about such techniques and their possible consequences. The more sophisticated user may wish to acquire the raw data and perform his own sampling and correction for purposes of incorporating the data with a particular geobase information system. Since inventory, monitoring, and change detection are expected to be among the principal applications of Landsat data by user agencies, it is important that the registration of this data with other data sets, or images, be well within one pixel. In addition, the user should be able to order small segments of data rather than a large standard frame. For example, many users work with areas corresponding to single or a few 7.5 minute quad sheets and much local processing time could be saved if they could obtain Landsat data covering only the area of their interest. To facilitate this type of archive retrieval it has been suggested that full geometric and radiometric calibration be performed by swath.

There are many other data features and processing considerations which could make the Landsat data more attractive and accessible to the potential user community. It would be extremely helpful if users were able to identify and have access to persons or facilitites in their local region who/which could provide the user with information and assistance in obtaining and using Landsat data. Many potential users today have the need for, and the interest in using, Landsat data but are limited in their hardware and software capabilities. There exists a need to provide modified versions of existing software packages which would fit on the smaller computers often available to these users. One of the most pressing needs today is the availability of a relatively inexpensive method of producing a permanent and attractive display of a processed final product. For most user agencies a black and white character printout is not an adequate final product. The user frequently desires a product in a color film format which is inexpensive and which preferably can be obtained locally.



Area Director:
Stephen J. Kohn
National Office of Ernst & Ernst
Cleveland, Ohio

Electronic fund transfer systems

Electronic funds transfer systems (EFTS) were considered for several years to present significant cost savings opportunities to financial institutions. Ideally, their wide scale introduction and acceptance would cut down paper work and enhance the market shares of those participating. Retailers offering services provided by financial institutions would also benefit directly by reduced check costs and possibly by attracting new customers through the convenience of banking facilities at their locations.

For several years, reports of varied systems designs, their promises for acceptance and the favorable impact on their sponsors were widely circulated. However, more recently the pattern of comments has become less optimistic. Now several key areas of concern are emerging:

- What are the roles of financial institutions and retailers as EFT alters the types of locations and of services which they can provide?
- What is the role of government in providing services, particularly when the services might not be a natural monopoly?
- What are the technological impacts of systems which allow for a high degree of concentration of information regarding an individual and his transactions.
- What legal modifications must be enacted due to the consumerist, anti-trust, and institutional changes which are brought about by EFT?

The last few years have demonstrated that the design and development of systems to achieve EFT's objectives may be the easiest part of EFT. Complexity arises because individuals and institutions are less willing to accept change than had originally been contemplated. Institutions often find more significant disruptions to their business activities than they had expected. Technological change is easier to design than to implement.

The National Commission on EFT, organized to review the societal ramifications of EFT, did not produce a report which could be easily implemented legislatively. Instead, there is still a significant polarization of interests coupled with diminished excitement about projects—most less successful than anyone had anticipated.

However, long term expectations are that certain phases of EFT still have significant promise:

- Automatic teller machine costs are declining rapidly and with more effective marketing, their installation will be more widespread.
- The automated clearing house had to achieve a stage of national interchange before it could truly have an impact on corporations. That has now been achieved.
- Designs of systems used at the point of sale still search for the formula which provides financial feasibility. Success here will be more difficult to achieve.

Meanwhile, financial institutions will explore other techniques, both technological, such as image retention and operational such as truncation, to reduce the costs of offering services. Retailers will make more extensive use of their internal point of sale systems for more economic transaction processing.

Systems auditability and control in an EFTS environment

by RUSSELL DEWEY
SRI International
Menlo Park, California

INTRODUCTION

Losses from accidental and intentional acts involving computers and data communications in financial institutions are growing. The current estimate of losses from credit card fraud alone is $500 million and could rise to $6 billion to $10 billion by 1986. Of greater concern than this problem is the growing potential for single instances of massive losses as EFTS grows and participants become highly dependent on continuously available computer services where most of their assets are stored in electronic form.

There is obviously a need to apply the principles of secure systems to the emerging development of EFTS, and much work has already been completed and published (See References 1 through 9). Most of this work has been focused primarily on the technology of security, such as: Personal Identification Numbers, Plastic Card Security, Cryptography, Physical Security, and Operating System Integrity.

On the other hand, there is also a growing awareness that EFTS security is a "people" problem. The risks, threats, and vulnerabilities to EDP systems derive primarily from the activities of individuals, either accidental or deliberate.[7,10] In the EFTS environment, individuals may be employees of the financial institutions, merchants, telephone company, hardware and software vendors, maintenance personnel, computer service bureaus, security personnel, and auditors.

Between the two orientations—technology and occupation—a body of information is emerging regarding audit techniques, and system application controls that, if implemented, could help reduce the potential for significant loss.[5] The purpose of this paper is to review and summarize the state of that art within the context of the EFTS environment.

THE EFTS ENVIRONMENT

The number of financial terminals being installed in remote locations to automate all or part of the transfer of credits and debits is increasing. By 1980 there may be over 100,000 terminals providing a variety of EFTS services, including:

- Deposits
- Withdrawls
- Transfers of balances between accounts
- Direct Debits for purchases
- Balance inquiries
- Check authorization and guarantee
- Credit card authorization and data capture
- Corporate cash management
- Funds concentration
- Corporate to corporate wire transfers

During the last few years, components of EFTS technology have begun to be implemented in a variety of different configurations, including shared access networks. A shared access network is one which allows for the switching of EFTS transactions to more than one possible destination, regardless of ownership considerations. Since this environment is more complex than single institutions dedicated to EFTS, we will focus on shared access networks for this paper.

The likely evolution of such shared access networks may be described by its principal components:

1. Remote Terminal—The terminal may be operated entirely by one person, such as an Automated Teller Machine (ATM), check guarantee terminal or cash management terminal. They may also be operated by an intermediary, such as a financial institution teller or merchant sales person.
2. Communications—The terminal may be connected to a computer located at a merchant, corporation, or potentially a government facility. In this case, the originating computer must be connected to the destination computer through intermediate computers and telephone communications facilities. Depending on the complexity of the local environment, the number of institutions participating, and economic considerations, the terminal may optionally be connected directly to a local financial institution, directly to a joint

venture shared EFTS switch, or through common carrier facilities directly to the destination.

3. Files—EFTS files take several forms and may be found in several discrete locations:

   - At the remote operator (merchant, corporation, government agency) there may be audit trails of EFTS transactions passing through, or decision parameters to control the processing of transactions when the remainder of the network is down. There are likely no balances, account data, or financial institution programs.
   - At the acquiring financial institution there may also be audit trails, but also balances and account data for its merchants and corporate customers and for transactions that do not need to be switched, such as an "on-us" debit. The acquiring financial institution may also have decision parameters to control the processing of "not-on-us" transactions when the remainder of the network is down.
   - At the EFTS switch there may be audit trails and decision parameters for any destination facility that may be down. There are likely no financial files, other than reconciliation totals and settlement amounts between institutions.
   - At the destination financial institution there may be audit trails, memo-post master file balances, transaction files with backup and off-line master files with back-up.

4. Communication Equipment—Each computer site will have specialized hardware to interface the EDP system to the external communication lines.
5. EFTS Software (Program Logic)—Each computer that is expected to participate in such an EFTS network must have specialized programs developed. These include:

   - Terminal protocols
   - Message format conversions
   - Switching and routing logic
   - Interface logic/protocol to other computers
   - Interface to existing financial software, such as:

     — Demand deposit accounting
     — Savings account accounting
     — Customer information data bases
     — New account processing

   - Specialized audit techniques and application controls

EFTS APPLICATION AUDIT TOOLS AND TECHNIQUES

The transition from traditional financial institution recordkeeping to today's EFTS application systems—characterized by large on-line master files, lack of manual intervention, and remote terminal entry—has brought with it design concepts that cannot rely on traditional manual control procedures. The accuracy and reliability of EFTS application processing is becoming more dependent on the incorporation of automated application controls (discussed in the next section). The purpose of the EFTS audit tools and techniques is to evaluate those controls, verify processing accuracy and continued compliance with processing procedures.

Some of the key audit tools and techniques that apply to the EFTS environment and verify the correctness of processing logic and controls include the following:

- *Base Case System Evaluation*—Execute application programs against test transaction data, entered through a "test mode" EFTS terminal, and compare results against pre-determined test results.
- *Parallel operations*—Execute new or revised application programs and existing application programs and compare results.
- *Integrated Test Facility (ITF)*—Enter test transaction data through live terminals commingled with live transaction data.
- *Parallel Simulation*—Live transactions are copied and processed against auditor programs that simulate processing logic. The simulation results are then compared to the live results.
- *Transaction Selection*—Systematically screening and selecting transaction samples entered through EFTS terminals for subsequent manual verification.
- *Embedded Audit Data Collection* (Sometimes known as *System Control Audit Review File: SCARF)*—Audit subroutines are embedded in the application programs to screen and select internal generated messages between EFTS logic modules that result from the original terminal transaction.
- *Terminal Audit Software*—Direct access to inspect live files during actual operation of the EFTS. The auditor will want to examine terminal polling lists, routing tables, merchant codes, floor limits, and default authorization tables.
- *Snapshot and Trace*—Secure documentary evidence of logic paths, control conditions, and processing sequence of a specific transaction. This is done by continuously recording transaction status for some selected transaction as it passes through the system.
- *Job Accounting Data Analysis*—Select, extract, and display job accounting data to monitor access to sensitive data files and on-line libraries.
- *Code Comparison*—Use of an off-line program to compare two versions of an application program to identify differences in coding.

EFTS APPLICATION SYSTEM CONTROLS

The purpose of the EFTS Application System Controls are to assure the accuracy and completeness of the processing results, the security of the environment in which the EFTS transaction is effected, and the effectiveness of the overall computer design and operations.

The controls described in the following sections are some of those likely to be beneficial in the remote terminal EFTS environment. They are grouped according to five phases:

- Transaction Origination and Data Entry
- Data Communications
- Computer Processing
- Data Storage and Retrieval
- Output Processing

TRANSACTION ORIGINATION AND DATA ENTRY CONTROLS

In developing controls within this phase, an organization may wish to implement:

- *Special-Purpose Forms*—At merchant operated or teller operated terminal locations. Proper form design encourages completeness and accuracy of data. In the cardholder operated terminal environment, Video Display Units (VDU) may be used for pre-formatted and/or interactive input.
- *Transaction Identification Cross-Reference and Source Document Numbers*—This control calls for sequentially assigned source document numbers (or sequential screen numbers) which are transmitted to the EFT processor as part of the transaction identification.
- *Sequence Log*—An internal log of which sequence numbers have been assigned to which merchant/teller locations or cardholder operated terminal location.
- *Signatures*—In the EFTS environment this control consists of appending a series of endorsements to each transaction as an audit trail of each node that handles it. It also serves as a "return destination" for undeliverable messages. The first endorsement is the terminal I.D.
- *User Identification*—A unique identification by class of user (cardholder, merchant, teller, bank supervisor) used to restrict the transaction to be processed as well as files that may be accessed.
- *Batch Serial Numbers*—Batches are identified by serial number to provide accountability of data, and to assist in the isolation of errors when an EFTS terminal cannot successfully reconcile at end of day (or other cut-off time).
- *Limit the Number of Transactions in a Batch*—This is done to simplify the reconciliation process.
- *Turn-Around Documents*—The source documents contain pre-recorded data in machine readable format. This will simplify processing in the event the on-line system is down and paper documents are used as back-up.
- *Retention Dates on All Transaction Logs*—This is based on legal requirements and management policy.
- *Source Documents Maintained at Origin*—In the EFTS interchange environment this means two things: (1) The paper documents (credit card vouchers, debit card vouchers, deposit slips, withdrawal slips) are maintained as closely as possible to the acquiring bank, to prevent unnecessary risk of loss through transportation, and (2) the electronic version of the EFTS transaction is also maintained at the acquiring facility until balanced at the end of the day. Subsequent settlement between banks, and actual movement of value data for posting can occur in batch mode at lower cost and higher reliability.
- *Error Logging*—Maintained by the acquiring facility to record and monitor the type of errors occurring at the terminal. Suspicious patterns of such errors could imply an attempt by an outsider to breech system security.
- *Verification of Re-Entered Data*—The data fields on resubmitted transactions are subjected to the same verification procedures as the original transaction.
- *Security of EFTS Terminals*—EFTS data entry terminals should be physically secure by placement in a lockable room, putting a keylock on the terminal itself, or by placing a lockable cover over the terminal device when not in use.
- *Terminal Logs*—Journals in the terminals to record all transactions.
- *Terminal Control Logs*—Journals in the terminal controllers to identify imposter terminals on the network (i.e. controller total of transactions should equal the sum of the legal terminals.)
- *Built-In Terminal and Terminal Controller I.D.'s*—These devices are provided with electronic identification that can be queried by the computer; used to validate proper terminal authorization.
- *Editing and Validating Routines*—These may be partially performed in the terminal itself, given anticipated hardware capabilities.
- *Passwords*—Used to verify that the input is being received from an authorized source. Likely to be the Personal Identification Number (PIN), although other techniques are being actively explored.
- *Unauthorized Access Attempts*—An immediate report is produced of unauthorized attempts to access the system. After a threshold of repeated attempts the system shuts down the terminal in question and allows access from that terminal only after intervention by security personnel.

DATA COMMUNICATION CONTROLS

In developing controls within this phase, an organization may wish to implement:

- *Secure Phone Equipment Rooms*—Locks and alarms are used to control access.
- *Network Configuration Polling Table*—No open addresses for unauthorized terminals to gain access.
- *Communication System Control Log*—To detect any unauthorized changes to the network done through network supervisor terminals.
- *Communication Line Routing*—Data communication lines are not put through public switchboards (PBX).
- *Local Loop Security*—Between the terminal, or termi-

nal controller, or computer, and the telephone company branch office.

- *Encryption Techniques*—Such as the National Bureau of Standards (NBS) Algorithm. For an excellent discussion of encryption techniques, see Kaufman and Auerbach.[1]
- *Forward Error Correction (FEC) and Automatic Request for Retransmission (ARQ)*—Common types of link control; to control transmission errors on each segment of a transaction path through the EFTS network to the cardholder data base and back to the terminal.
- *Message Sequence Number*—Each transaction at an EFTS terminal may generate several messages (terminal-to-acquiring CPU, acquiring CPU-to-switch, switch-to-cardholder CPU, etc.). This provides a traceable log to match inquiry with response, detect lost messages, or detect imposter messages.

COMPUTER PROCESSING CONTROLS

In developing controls within this phase, an organization may wish to implement:

- *Monitoring of Internal-Generated Messages*—Internal generated messages (such as an authorization request to the issuer on an interchange withdrawal transaction) should be uniquely identified and cross referenced to the external transaction that spawned them.
- *Control Totals*—Reasonableness checks and internal control totals between program modules.
- *Default Option*—Each level in the system hierarchy may need to make decisions in default when the rest of the system is down, or when it is uneconomical to do so (e.g., the merchant bank may authorize a $3.00 transaction against a negative file instead of requesting authorization from the issuer).
- *Dual Fields*—All entries should be carried as a credit against one account and a corresponding debit against another, throughout the life of the transaction.
- *Arithmetic Accuracy*—Techniques such as double arithmetic and arithmetic overflow checks are placed in critical points in the application.
- *Destructive Update*—Debit and credit type entries are used to correct error conditions, not delete or erase commands.

DATA STORAGE AND RETRIEVAL CONTROLS

In developing controls within this phase, an organization may wish to implement:

- *File Classification*—Each file is classified by security level, and access is restricted by level.
- *Data Base Control Table*—No data base access is allowed unless it comes from an authorized program module.
- *Program Linkage Control Table*—These tables control the authorized module linkage between programs.
- *Dormant Files/Accounts*—The system reports on dormant files/accounts that suddenly have activity on them. Many attempts at EFTS fraud try to make use of dormant accounts.
- *Excessive Activity*—The system reports on records and data fields that have excess activity over a certain threshold. It could mean that a lost or stolen card is being used.

OUTPUT PROCESSING CONTROLS

In developing controls within this phase, an organization may wish to implement:

- *Reconciliation*—The response to an EFTS request is matched up and reconciled with the original request before completing the transaction.
- *Transaction Log*—The central transaction log is periodically matched against the journal tape in the EFTS terminal. These totals are also verified against individual application control totals.
- *Output Activity Review*—Real-time statistics such as the number of terminals on-line, transaction quantities, and circuit traffic are reviewed by the appropriate management. System generated subjective judgments, such as default authorization, are reviewed by user departments.
- *Device Verification*—EFTS devices whose action such as imprinting a card or dispensing cash is controlled by the application programs will send a status report back to the host computer to verify completion of the requested mechanical operation.

CONCLUSION

Any remote terminal-based interactive system will face a variety of threats. EFTS, in particular, because of the electronic movement of funds and value, will be particularly vulnerable. Fortunately, a number of system audit techniques and application controls are available to the system designer; and more are under development. Depending on the particular system design, careful implementation of these audit techniques and application controls is likely to diminish the vulnerability to those threats.

REFERENCES

1. Kaufman, D. and K. Auerbach, "A Secure National System for Electronic Funds Transfer," *Proceedings 1976 NCC*, pp. 129-138.
2. Mazzetti, Joseph P., "Design Considerations for Electronic Funds Transfer Switch System Development," *Proceedings 1976 NCC*, pp. 139-146.
3. Backman, Frank, "Are Computers Ready for the Checkless Society?," *Proceedings 1976 NCC*, pp. 147-156.
4. "The Analysis of Certain Threats to EFT System Sanctity," Kranzley & Co., Cherry Hill, N.J.—A study done for the Electronic Industries Foundation; January 1977.

5. "Systems Auditability and Control Study," SRI International, January 1977. (Available from the Institute of Internal Auditors, Orlando, Florida)
6. "Introduction to EFT Security," prepared by the Division of Management Systems and Economic Analysis, FDIC, Washington, D.C., August 1976.
7. Parker, Donn, *Crime By Computer,* Charles Scribner's Sons, New York, 1976.
8. Martin, James, *Security, Accuracy and Privacy in Computer Systems,* Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
9. Hoffman, Lance J. *Security and Privacy in Computer Systems,* Melville Publishing Company, Los Angeles, 1973.
10. "A Guide to EDP and EFTS Security Based on Occupations," A report prepared for the FDIC by SRI International, Menlo Park, California, 1977.

Electronic fund transfer systems and quality of life

by ROB KLING
Information and Computer Science
University of California
Irvine, California

SOCIAL MEANINGS OF ELECTRONIC FUNDS TRANSFER SYSTEMS[1]

Electronic funds transfer technologies are rapidly emerging. The many different technologies, collectively called EFTs, provide for the transfer of credits and debits and for a variety of related information services. They are likely to be widely used and may easily penetrate our everyday lives. If they become pervasive, they can alter the texture of our society with an intensity similar to that of the radio, automobile, jet plane, electricity, telephone, and television.

In their early years, new technologies offer tremendous promises to increase the quality of life for their consumers. However, when high technologies are adopted on a large scale, they often lead to social problems as small problems magnify in size. For example, cars were an optional novelty and recreational vehicle in the horse and buggy Los Angeles of 1900. Today they dominate the social architecture through freeways which link sprawling suburban tracts, regional shopping centers, and widely scattered workplaces.

Most complaints about modern "industrial technology" focus on large scale, relatively inflexible technologies with centrally managed infrastructures. Thus, the automobile, television, jet planes, and nuclear power raise troublesome problems. On the other hand, individually controllable technologies such as the telephone, cameras, stereo systems, pocket calculators, sewing machines, electric typewriters, and photocopiers have been applauded or silently accepted. It is not "technology" per se, but the social institutions that develop around a technology, that result in praise or mistrust and complaint.

Research studies can shed light on the impacts of different EFT arrangements. But investigators make important assumptions when they select particular social patterns and values to study. It is easiest to select "dominant social trends" such as increasing affluence, urbanization, market concentration, and levels of social integration which have characterized American life during the postwar years. However, the criteria for selecting some trends rather than others are rarely made explicit. *Which* trends will characterize our futures often lie in the eye of the beholder.

Trend analyses often obscure the possibility of important social choices. If the banking industry has been marked by increasing concentration in the last two decades, need that continue? Is it necessary or desirable that all EFT arrangements support that "trend"? While most EFT arrangements are promoted in a way which is consistent with a consumption-oriented growth economy, are any EFT arrangements particularly consistent with a steady-state economy?[2] Such questions are hard to ask, let alone answer with casual trend analyses. This paper provides an alternative form of analysis by explicitly using a theory of social life and a design space for EFT arrangements.

QUALITY OF LIFE

We experience our social worlds and define the quality of our lives in very concrete terms: enjoying coffee with a friend, being frustrated over the noise of a radio or plane, or being disappointed in love, tense over the prospects of losing a job, or amused while watching a child play with a cat. "Quality of life" denotes the subjective meanings we place on our experiences and opportunities. In general, analysts define a good quality of life both by expressed personal satisfactions of people and by the absence of discernible social problems such as crime and unemployment.

Although, there are growing bodies of empirical research about perceived quality of life,[3] *theories* of "quality of life" are rare. Gerson,[4] however, has developed a position which merits serious attention. It emphasizes the relative power of participants in social settings as well as their relative social resources and satisfactions. Gerson's theory forms the basis of the analysis presented here.

We begin with a concept of social order which develops in and through a process of ongoing negotiation.[5] In Gerson's perspective, the problem of reconciling the good of individuals with the good of the largest social group in which they act is that of "creating and managing a pattern of negotiations which is viable throughout the range of (social) scales at which it takes place."[4]

This conception of social life emphasizes joint participation in multiple settings. The subjects of negotiation in each setting are flows of resources and constraints upon them. Resources include both *material goods* such as time and money, and *social goods* such as skills and sentiment (pride, embarrassment, state of mind, etc.). The quality of life of a

participant in a specific setting increases with:

- the resources at his disposal;
- his ability to negotiate new resources.

EFT arrangements may influence the quality of life for participants of a social setting on two different levels. First, they may influence the resources available to *individuals in everyday life* and the ability of *individuals* to negotiate with each other and with organized groups. Secondly, they may alter the resources available to specific organized *groups* and influence their ability to negotiate resources and commitments. In the next section, we will develop six categories for describing patterns of resource distribution and negotiating ability.

CHARACTERIZING QUALITY OF LIFE

We need a systematic way to think about the influences of alternative EFT arrangements upon the *quality* of social and institutional life. We emphasize six characteristics of social settings.

Resource Distribution

- *Opportunity*: the kinds of resources that an EFT arrangement adds to a particular social setting.
- *Ecology*: The extent to which specific EFT arrangements save or consume disproportionately large amounts of scarce non-renewable resources.
- *Coordination*: The extent to which specific EFT arrangements alter the ease with which a party can manage his activities across all the settings in which he participates.

Negotiating Ability

- *Intelligibility*: The extent to which social, organizational, or technical arrangements can be understood by the participants in a specific setting.
- *Dependency*: The extent to which the participants in a specific setting can choose to use or not use specific EFT arrangements without bearing "excessive" costs in resources such as money, time, sentiment, etc.
- *Agency*: The extent to which participants in a particular setting can influence the social, organizational, or technical arrangements in force.

In an ideal world, EFT arrangements would increase opportunities for all parties, and be ecologically efficient; relative to current arrangements, they would be more intelligible, increase all party's sense of agency, and generate so little dependency that consumers could choose to get on and off with negligible costs. In actuality, these values trade-off against each other.

This paper analyzes the ways in which alternative EFT arrangements influence the six aspects of negotiated social life. However, first we need to characterize different EFT arrangements.

CHARACTERIZING EFT TECHNOLOGIES

Most analyses of EFTs are developed in terms of either specific EFT arrangements (e.g., point-of-sale networks, automatic tellers) or specific services (e.g., debit cards, preauthorized payments).[6–10] However, for shaping public policy, it is important to be able to *distinguish between* EFT arrangements (technologies and their associated social organization) that have different abstract features.[7] For example:

- A point-of-sale (POS) network which serves the major department stores to help them improve their cash management may be more effective if it is national rather than regional in scope;
- A national POS network may be much more convenient for people who travel cross country than would a set of regional networks;
- The disruption caused by an unreliable POS network would be less severe if it were regional in scale rather than national.

Many discussions of the opportunities offered and problems posed by different EFT arrangements are *reasoned* through in terms of certain properties of EFTs. For example:

- Discussions of the accessibility of EFT arrangements to smaller banks focus on different ways of *sharing* facilities;[9]
- The impacts of different EFT arrangements on personal privacy are discussed in terms of the *richness* of the data they contain and the extent to which people access such systems;[11]

Analyzing alternative EFT arrangements can be facilitated by developing an appropriate set of attributes to characterize them. In this analysis we propose seven "internal" features:

- *versatility*: denotes the variety of services provided with a particular EFT arrangement.
- *scale*: denotes the number of points of contact and people served by a particular EFT arrangement.
- *coupling*: denotes the extent to which EFTs that provide separate classes of service or which could be factored to serve separate geographical regions are joined together.
- *reliability*: denotes the extent to which specific EFT arrangements are trustworthy, e.g., that the equipment will not fail and that data or funds will not "leak" unexpectedly from the system.
- *richness*: denotes the volume and variety of data in a specific EFT that can be linked to a particular account holder or class of transaction.
- *accessibility*: denotes the ease with which people or organized groups may gain access to the services or

data which are collected or provided by a specific EFT arrangement.
- *production costs*: denote the costs of fabricating, administering, operating, and maintaining a particular EFT arrangement.

These features can characterize a variety of EFT arrangements. For example, under current arrangements, automatic teller machines provide a few specialized services, are not very versatile, operate on a small scale, are not highly coupled, are relatively reliable, are data poor, and are relatively accessible to bank customers. In contrast, national automated check processing networks would be more versatile, operate on a large scale, may or may not be coupled with other services, are of uncertain reliability, would be data rich and very accessible.

This set of characteristics can be used to compose many important characteristics of EFTs. For example, convenience increases with ease of access, reliability, and versatility. Similarly, Rule's[11] concept of "surveillance capacity" combines our concepts of *richness, scale,* and *accessibility*. These features are not completely independent. For example, the *coupling* of two diverse EFT operations (such as a POS network and a credit-card transmission system) could lead to a more *versatile* system.

These "internal" features characterize EFTs independently from the world in which they reside. We also need to consider the "fit" between specific EFTs and the social, legal, and economic world in which they are embedded. For example, automated checking systems will be substantially more convenient if electronic records are accepted as proof of payment. This motivates our sole "external" characteristic:

- *fit*: an EFT arrangement has a better fit with a given social, legal, political, or economic arrangement to the extent that the conventions of the two coincide.

ASSESSING EFT IMPACTS

The last two sections developed a design space for EFTs and a specific set of features to describe the quality of life in specific social settings. The analyses developed here will be clustered under each of the six aspects of quality of life developed earlier: opportunity, intelligibility, dependency, agency, ecology, and coordination. Each of these six aspects will be considered from the perspectives of (1) everyday life of individuals and (2) institutional arrangements.

Opportunity

Increases in "opportunity" are the raison d'etre of EFT developments. While EFTs may increase opportunities for some people, they may diminish opportunities for others. Presumably, many individuals would gain access to new financial and related services.[10] Yet, some EFT arrangements might endanger the existence of unpopular political groups by enabling abusive surveillance and harassment.[6,7]

Most EFTs have been developed by private businesses which seek to open new markets and increase operating efficiencies. And they have been supported by public agencies such as the Federal Reserve Board which seek to enhance their institutional efficiency and effectiveness. Since most accounts of EFT arrangements emphasize the enhanced opportunities the technology provides, they will not be elaborated here.[9,10,12] In general, EFT arrangements will offer greater opportunity to larger institutions with increasing levels of *scale, coupling, diversity,* etc.

Intelligibility

The National Commission on Electronic Fund Transfers argues that EFT developments should be fostered so as to maximize the number of available kinds of EFT-based services.[9] Yet, different EFT arrangements can alter the coherence of people's business relations in two primary ways:

- They can simplify or complicate specific transactions and financial contracts;
- They can alter the demands made upon people to understand available products.

As personal choices increase, so do the demands upon them for understanding the array of options available and the relative benefits of each. We have little understanding of the conditions under which people experience increased choice as bewildering.

Some leads may come from research on human memory. Twenty years ago, George Miller publicized his famous finding that people could retain "seven plus or minus two" chunks of information in short term memory; the number of distinct things that people can remember and compare depends upon the ways in which their characteristics are coded. It also appears that individuals cannot easily remember or cope with the details of an arbitrarily large number of choices.[13] In economic terms, the "information costs" borne by consumers may increase as additional EFT arrangements become available.

A second issue is the complexity of particular financial transactions. Altering the complexity of a few routine transactions, such as paying a utility bill, has minor ramifications. In this case, authorized pre-payment schemes may have mixed effects: they may simplify routine payments, but make it more difficult for consumers to detect and correct foul-ups. However, if EFT arrangements become so widespread that they become the medium for most routine payments for essential services, then individual consumers may find the technology not altogether "labor or attention saving."

One societal arrangement is preferable to another *if* it can be easily comprehended by lay people as well as by specialists. Versatile EFT arrangements which provide a myriad of related information services (such as funds transfer, accounting, electronic mail, etc.) to a variety of institutions

can easily boggle the minds of most people. Large scale EFT arrangements should be judged by a criterion of "manageable complexity." This means both:

- that people can understand the contours of an EFT system;
- that EFT arrangements should not confuse the regulatory distinctions between banking, communications, etc. and the domains of the regulatory agencies such as the FCC, FTC, etc.

EFTs become less coherent as they increase in versatility, scale, and coupling. Precisely those features that increase the "opportunities" provided by EFTs to some parties can render them incomprehensible to the people who are authorized to act on behalf of the public.

"Intelligibility" is also influenced by purely technical issues. As Weizenbaum[14] points out, very large computer programs develop in such a way that they cannot be easily understood by their designers or implementors. This issue has many difficult aspects. Some concern the limited ways we have of representing the design and behavior of computer programs. Users and auditors of computer programs have occasion to ask very different questions about a set of computer programs and the hardware on which they run:

- what computations (tasks) can these programs perform?
- What kinds of computing resources will they consume in the course of carrying out a specific computation?
- Under what conditions will these programs fail to operate properly?
- Under what conditions will data "leak" from one path in these programs to another path in these programs?

Currently, computer specialists have very few means of representing system specifications. These difficulties are compounded by the ways in which large software systems are developed; on such projects, many architectural details which are unspecified in a grand design are resolved (or altered) by specialists who are working on complicated subproblems. The net effect is that it is common for many programmers to introduce minor but significant design details which are essentially undocumented.

The quality of available documentation is also strongly influenced by the widespread distaste many professional programmers feel for documenting their work. While different strategies for organizing software production and document preparation are being developed to help minimize these difficulties, there are, in general, no clear and reliable solutions today.

Dependency

To the extent that people depend upon a particular arrangement, they may lose their power to negotiate alternatives later on. Benign technologies provide people discretion in their decisions either to use them or drop them should they choose to.

Dependency means both reliance upon a particular good or service or reliance upon a single or small set of suppliers. While all social life is based on complex patterns of mutual dependency, "good" social relations allow some parity between parties.

New technologies such as computing, in general, and EFTs, in particular, attract many consumers because they extend their "reach." The same can be said for electricity, automobiles, and the telephone. Problems emerge when we develop complex forms of social organization which are dependent upon the reach provided by a single technology or system. Thus, the Northeast power failure of 1965 and the New York failure of 1977 were particularly disruptive because people had few alternative energy sources for illumination and appliances.

Not all pervasive technologies are essential. Plastic wrap could disappear, or even television could be discontinued without a major national emergency. However, we would have a tougher time cutting back telephones, electricity, and automobiles. To the extent that EFT services *replace* current means of payment, they will become more like electricity than like plastic wrap.

Particular individuals may organize their lives so that they are more or less dependent upon a given technology. Thus, in Southern California, some individuals insist on driving their cars even one city block, while others make elaborate arrangements so they can bicycle to work amidst a maze of freeways and boulevards. It is a quite different question whether a social collectivity can easily switch from one technology to another.

Payments mechanisms are an essential part of a social infrastructure. In that, they are like transportation and personal communication. Once an attractive new technologically based capability is developed, many institutions can alter their style of business so that they effectively depend upon it.

Large scale, specialized, complex technologies take a long time to plan and implement. In the course of developing a plan, thousands of people may become involved. Thus, even "feasibility plans" develop tremendous inertia. Even when major design flaws are found, it is difficult to redesign or retrofit large scale systems. The BART system in the San Francisco Bay Area provides a particular object lesson here.[15]

In general, the larger the *scale, versatility,* and *accessibility* and the less *costly* an EFT service becomes, the more likely we are to become dependent upon it.

Agency

In the western liberal tradition, personal influence is a central moral and political value. In the American political system, in theory, the voter is sovereign. And in the neoclassical theory of the free enterprise system, the consumer is held to be sovereign.[16] Social and technical arrangements should be valued to the extent that they enhance the sovereignty of individuals. To the extent that they enhance the

influence of already powerful groups over weaker groups, they should be viewed with caution.

We have little understanding about the ways in which different EFT arrangements alter the influence of people in their daily dealings with business enterprises and government agencies. EFT technologies can enhance the control of either individuals or EFT-using institutions. The actual patterns of control that emerge have less to do with technical capability than with institutional prerogatives. For example, while it is easy to implement "stop payment" mechanisms in automated check processing systems,[6] proposals for such systems usually neglect these features. In fact, studies of consumer preferences seem to show that most people who use some combination of cash, checks and credit cards are satisfied with the sense of control over their finances that these payment media offer.[9]

One may wonder whether particular EFT arrangements alter the ability of the polity to act as a collective agent on its own behalf. The recent court cases brought by the U.S. Department of Justice against IBM indicate that it is easy for a corporate giant to overwhelm the resources of a federal agency. Specific EFT arrangements may lead to an even weaker balance of power between Federal or state regulatory agencies and large EFT using institutions. This may happen by:

- certain EFT arrangements fostering further concentration of capital in selected sectors of the economy.[12,17] The emergence of "superbanks" would be one such development. While there are over 14,000 commercial banks in the U.S., over 70 percent of all deposits are controlled by the 100 largest banks.
- EFT arrangements becoming so complex that EFT operations could not be easily influenced by focused collective action such as Federal legislation or regulation.

These observations don't lead to easy policy positions since federal regulatory agencies have also been an imperfect device. They have been criticized for both insensitivity to the public, and for being whimsical agents for businesses to deal with.[17,18] In general, the larger the scale, the more versatile and the more highly coupled EFT operations become, the more they are likely to foster large enterprises on one hand, and demands for Byzantine regulatory devices on the other.

Ecology

In a period of limited resources, we must emphasize technologies which are energy and resource efficient. Projections about energy and resource efficiency and conservation have to be carefully assessed for specific technologies in specific settings. For example, narrow estimates of the paper-saving efficiency of photocopiers would focus on the amount of carbon paper saved. However, in most settings where copiers are used, paper and energy consumption has probably increased. This is not to say that photocopiers ought to be restricted; rather that projections for resource conservation need to be made based upon expected volumes of service rather than upon efficiency criteria alone.

On a larger societal scale, the resources consumed by particular EFT arrangements must include the costs of capital construction as well as the resources to operate them. In a recent analysis of the economics of mass transit, Lave[19] has argued that rail systems have lower *operating costs* than automobiles and buses; However, they have *higher overall costs* when one factors in the costs of constructing rails and vehicles. Is it possible (or likely) that the same patterns hold for large-scale EFT arrangements relative to their manual (paper-based) precursors?

An ironic development with some current EFT arrangements used by banks is that they are used to increase the volume of financial transactions generally.[10,20] Thus, a current side-effect of some "paperless" EFTs is a net increase in the net amount of paper flowing through the banking system.

If energy prices continue to rise, the relative attractiveness of EFT based services and its traditional alternatives will shift. If particular EFT arrangements are clearly conservative of resources, they might be promoted through positive public policy. On the other hand, those EFT arrangements that are relatively expensive in energy and nonrenewable resources might be discouraged.

Coordination

It is an open question whether specific EFT arrangements increase or decrease the ease with which users and others can manage their social and business relations. It is a commonplace observation that "labor saving" devices seem to have exacerbated the plight of the "harried leisure class."[21] While EFT advocates proclaim the array of "conveniences" that EFT arrangements may provide, these claims seem to be oddly isolated from the overall problems that they may induce. When they work well, some EFTs may lead to small increases in convenience for consumers. In addition, new information services, such as accounting-like payment summaries, may help people organize their business affairs. However, when payments records are in error, the "costs of coordination," e.g., correcting errors, may increase. Generally, people can spend less effort coordinating their lives in a society that depends upon a large variety of EFT-related services to the extent that they lead to greater *intelligibility*, greater *agency*, and less *dependency* for people.

In a recent article, Rose[12] suggests that small and large businesses may be able to increase their fiscal control through bank-related cash-management services. In theory, the same characteristics of EFTs that lead to lower costs of coordination for individuals and larger institutions should be identical. In practice, the larger institutions can "purchase" greater expertise than smaller businesses and individuals. Thus, we expect larger institutions to gain more from EFT-based information services when they work well. However, organizations, large and small, will ease their burdens of coordination to the extent that EFTs grow in *versatility, richness,* and *reliability*. In addition, enterprises with geo-

graphically dispersed activities—already a large and growing fraction of private enterprise and public agencies—will benefit from increases in the *scale* of EFT arrangements.

EFTs and broader social patterns

This paper has developed a design space for analyzing different EFT arrangements. It appears that certain EFT designs (e.g., *smaller scale, less coupled*) present fewer potential social problems that EFTs of alternative design.

Despite these normative considerations, EFT developments are rapidly emerging in the current industrial system[22] and the existing legal and regulatory arrangements. Growth is a dominant feature of large enterprises; both the computer and financial industries are highly concentrated.[23,24] For expansion seeking suppliers of computing equipment, EFTs offer a lucrative new market. For financial institutions, EFTs offer strong potential for increasing their market share. Firms in both industries may gain from large scale, rich, and versatile EFTs.

As EFT services spread, failing to integrate several related services that are manually linked by different users or consumers may appear oddly "inefficient." Thus, the integration of large scale, highly versatile EFTs may be less the product of a grand design than the by-product of many small "locally rational" decisions. Unfortunately, the "tyranny of small decisions" easily leads to an overall arrangement which no one would have sought had the long run designs been made explicit early on.[25]

It is commonplace to notice that some values trade off, that others are mutually supportive, and that some design values and social values conflict. More reliable technical systems often have higher production costs. Larger scale and more versatile EFTs which provide more opportunities for users and consumers. But they are also likely to be less intelligible and diminish agency when compared with less expansive EFTs.

Whether one speaks of value conflicts or "tradeoffs," the social choices we face are no longer very simple.[26] There is a cartoon caption that reads:

> There's a price tag on everything. You want a high standard of living, you settle for a low quality of life (quoted in reference 27).

In order to help make our choices less grim, I believe that we should have clear answers to several questions. These are elaborated in the next section.

RESEARCH AGENDA

The analyses of this paper indicate some of the social dilemmas posed by EFTs. EFTs are not all alike. Some, such as automated tellers, may be more amenable to small scale operations than, say, automated check processing.[6] To the extent that EFTs increase the intelligibility of social arrangements, ease of coordination, and social agency, while decreasing dependency for the public, they may be thought of as especially benign technologies. These hopes may be problematic at best and naive at worst. Studies which need to be done include those to:[28,29]

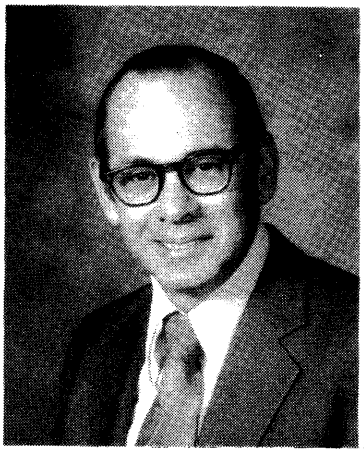
- Examine the market structure of firms which use and provide EFT services to determine which EFT arrangements foster increased concentration amongst a few firms and which arrangements support a diverse array of smaller scale businesses.
- Identify important social costs and develop ways to internalize them into pricing schemes for EFTs. Many of the social costs of EFTs suggested in this paper and elsewhere[6,7] are intangible. Furthermore, they may appear only after EFTs reach an advanced form of development and widespread use.
- Understand what it means for consumers to have "perfect information" about the choice of EFT services. If social costs cannot be internalized into EFT pricing, as is likely, determine how the public can be best informed about what they are.
- Develop stronger forms of consumer sovereignty in oligopolistic markets. It is easy to assume and difficult to demonstrate the sovereignty of consumers in complex business and financial markets. The structural role of consumers relative to EFT providing institutions ought to carefully investigated.
- Understand which EFT arrangements are most consistent with a low growth rather than high growth economy.
- Understand how to develop EFT technologies and their associated social arrangements so that they are easy to roll-back, if necessary, without major social upset.

Effective public policies to resolve many of the dilemmas raised by pending EFT developments must foster broad consensus among social groups with conflicting values and interests. In setting a research agenda, we should be fully aware that by selecting important values for study, we may superficially threaten active interests. This is a problem of doing serious research in a highly charged setting.

REFERENCES

1. For a more detailed account of the analyses presented here, see Kling, Rob "The Social and Institutional Meanings of Electronics Funds Transfer Systems," Information and Computer Science University of California-Irvine, Irvine, Ca.
2. Daly, Herman E., (ed.) *Towards a Steady-State Economy,* W. H. Freeman and Co., San Francisco, 1973.
3. Strumpel, Burkhard (ed.), *Economic Means for Human Needs: Social Indicators of Well-Being and Discontent,* Survey Research Center, Ann Arbor, Michigan, 1976.
4. Gerson, E., "On 'quality of life'," *American Sociological Review,* 1976, *41*(5), 793-808.
5. Strauss, A. L., *Negotiation: Order and Context,* Jossey Bass San Francisco (in press).
6. Kling, Rob "Passing the Digital Buck: Unresolved Social and Technical Problems in Electronics Funds Transfer Systems," Irvine, Ca: Public Policy Research Organization, WP-76-13, May 1976.

7. Kling, Rob "Value Conflicts and Social Choice in EFT Developments," *Communications of the ACM* (in press).
8. Balderston, F., E. J. Carmen, and A. Hoggatt, Computers in Banking and Marketing, *Science,* Vol. 195, March 18, 1977, pp. 1115-1119.
9. National Commission on Electronic Fund Transfers EFT in the United States, Washington, D.C., October 1977.
10. Arthur D. Little, The Consequences of Electronics Funds Transfer—A Technology Assessment of Movement Towards a Less Cash/Less Check Society, Report C-76397, Cambridge, MA., January 30, 1975.
11. Rule, J., *Private Lives and Public Surveillance: Social Control in the Computer Age*. Schochen Books, New York, 1974, pp. 269-277.
12. Rose, Sanford, "More Bang for the Buck: The Magic of Electronic Banking," *Fortune,* 1977, 95(5), pp. 212-226.
13. Winner, Langdon, "Complexity and the Limits of Human Understanding," in *Organized Social Complexity,* Todd La Porte (ed.) Princeton University Press, Princeton, N.J., 1975.
14. Weizenbaum, J., *Computer Power and Human Reason,* W. H. Freeman and Co., San Francisco, 1976.
15. Webber, M., The BART Experience—What Have We Learned? *The Public Interest,* No. 45, Fall 1976, pp. 79-108.
16. Mansfield, Edwin, *Microeconomics,* W. W. Norton & Co., New York, 1975.
17. Reid, S. R., *The New Industrial Order, Concentration Regulation and Public Policy,* McGraw-Hill, New York, 1976.
18. Noll, Roger, *Reforming Regulation,* The Brookings Institution Washington, D. C. 1971.
19. Lave, Charles, Rail Rapid Transit: The Modern Way to Waste Energy, *Transportation Research Record,* National Academy of Sciences, Washington, D. C., (in press).
20. Business Week, "A Retreat from the Cashless Society," April 19, 1977, pp. 80-90.
21. Linder, Stephan, *The Harried Leisure Class*. Columbia University Press, New York, 1970.
22. Kling, Rob and Elihu Gerson, "The Social Dynamics of Technical Change in the Computer World," *Symbolic Interaction,* 1(1), December 1977, pp. 132-146.
23. Eichner, Alfred, *The Megacorp and Oligopoly,* Cambridge University Press, London, 1976.
24. Galbraith, John R., *Economics and the Public Purpose,* New American Library, New York, 1975.
25. Rowley, Charles K., "Pollution and Public Policy," in Culyer, A. J. (ed.) *Economic Policies and Social Goals,* Martin Robertson and Co., London, 1974.
26. Kling, Rob, "Five Models for the Social Accountability of Computing," *IFIP TC-9 Newsletter* 2(1), December 1977.
27. Harman, W., *An Incomplete Guide to the Future,* San Francisco, CA; San Francisco Book Co., San Francisco, CA, 1976.
28. Kling, Rob, "The Social and Institutional Meanings of EFTs," A Research Agenda, Public Policy Research Organization University of California-Irvine, Irvine, CA., 1977.
29. King, J. and K. Kraemer, "Electronic Funds Transfer as a Subject in Technology, Society and Public Policy," *Telecommunications Policy,* 2(1) March 1978, pp. 13-21.



Area Director:
William G. Key
NCR Corporation
Cambridge, Ohio

Special purpose terminals

This technical area focuses on the current state of the art in Special Purpose Terminals and provides a forum for discussion of the future trends in Special Purpose Terminals. The sessions are balanced to provide sufficient definition, by contrasting the Special Purpose Terminal with the General Purpose Terminal, along with special aspects of terminal operations such as voice input.

The criteria associated with planning a special purpose terminal for a given marketplace is explored. The major influences exerted by Engineering, Marketing, Manufacturing and competition are examined and placed in perspective. It gains this perspective by examining basic trade-offs on cost, performance, implementation, marketplace, competitiveness and profitability.

The areas of market and product existence, product salability, and administrative analysis are examined with details on decision steps required to complete the product planning. It highlights the decision criteria and planning process by walking through a special purpose terminal that was designed to displace a general purpose terminal.

Making a special purpose terminal work in a general purpose system environment complements the decision to produce a special purpose terminal. Labor costs for factory operation along with the need to reduce inventory costs have been a strong motivator for the special terminal in the factory. Most factory oriented special purpose job oriented terminals need to directly communicate with large scale general purpose data processing systems. This technical area presents the system implementation used to connect an interactive factory data collection system on line to a large scale IBM data processing system operating in an IMS data base environment.

The factory terminal system has many requirements which include both general purpose and special purpose parameters: Time and Attendance (Special Purpose), Last Job Checkout (Special Purpose), Inquiry (General Purpose), and Material Tracking (General and Special Purpose).

The method chosen for interfacing the complete system was to emulate a general purpose network of IBM 3270 type CRT terminals. The emulation technique provides for many on-line different modes of operation including the following: 3270 Emulation in the remote communications mode, 3270 Emulation with direct channel attachment, Transparent mode of operation, Data collection

mode of operation, and Interaction with both main data base and fast response local files.

The future of Special Purpose Terminals is broadly covered with emphasis on input devices that complement the environment that we find in the special purpose terminal. "The Future of Special Purpose Terminals" examines current trends in computer terminal development and extrapolates these into 1978-1988 time period, with special emphasis on the transition from large scale integration (LSI) to very large scale integration (VLSI). This session offers projections concerning the demands for special purpose terminals and on the ability of the industry to respond to these demands. Additional explorations are examined in more detail by the panelists who have chosen to concentrate on specific technology areas as well as on the extent in which future advances in technology are likely to affect terminal systems presently in use.

One point of view that will be presented asserts that the single most important attribute of the interactive computer terminal of the future is that it contains a general-purpose computer with the approximate characteristics of a present-day mini-computer. Another point of view points out that voice data entry, long recognized as an ultimate step towards simplifying communications between a human and a machine, is now a reality. Over 200 voice terminals are currently in operation in a variety of industrial and commercial applications in eight countries around the world. New speech input terminals using microcomputers have been introduced which can replace and/or complement intelligent CRT/keyboard stations by enabling the user to enter data by voice.

The potential applications of voice data entry are determined by the economic advantages of voice input as compared to other alternative data entry devices and/or techniques. As the cost of voice data entry terminals decreases, more justifiable applications will arise, particularly when the true costs of data capture are considered. In some cases, voice input offers advantages which far outweigh the direct labor savings and permits certain operations to be achieved which could not easily be accomplished using alternative data input techniques.

A third point of view addresses the impacts of technology advances on current data entry terminals and their use. How long does it take for an incumbent industry or profession to yield to the onslaught of new advances? It observes that keyboard data entry is supposed to have been "dying" for over 20 years, ever since the introduction of optical character recognition (OCR). Each time a new device is developed the cry "keyboard data entry will become obsolete in five years" is heard again. Yet through all of this, there are more data entry keyboards of some type in use today, and more are projectd for the future.

Decision criteria in terminal product planning

by ERIC C. WESTERFELD
NCR Data Pathing, Inc.
Sunnyvale, California

THE SPECIAL PURPOSE TERMINAL VS. THE GENERAL PURPOSE TERMINAL

The concept of special and general purpose products is, to a great extent, in the mind of the end user. The application areas have the greatest impact on the definition of special purpose and general purpose. Products which address applications in many areas are considered general purpose. Also, products which address no specific application but can be adapted with additional software or hardware are considered general purpose.

The concept of the product incorporates the market or end usage of the product as special or general purpose also. A grid of special purpose and general purpose terminals and special purpose and general purpose applications emerges (Figure 1). In reality, any attempt to make clear line distinctions as shown in Figure 1 are not realistic, and the more general grid in Figure 2 shows the complexity of this graphic determination. For presentation purposes, only Figure 1 presentation will be used to denote:

A—A general purpose terminal used in a general purpose application.
B—A general purpose terminal used in a special purpose application.
C—A special purpose terminal used in a general purpose application.
D—A special purpose terminal used in a special purpose application.

At this level, some generalities can be abstracted, and some limits to the directions for this paper can be made.

First of all, the general purpose application with general purpose equipment (A) is the realm of the computer center or system user, where the manufacturers equipment or OEM equipment predominates. As a general application evolves, usually additional software is generated, and additional equipment is purchased. Special purpose terminals are difficult to support because of the increased software burden compared to manufacturer supplied system interface oriented terminals (Section C). This is an area in which the special terminals do not survive well. For the purpose of this paper, Section A will not be pursued again.

Section C, which shows special purpose terminals used in general applications, was briefly mentioned above. The special purpose terminals do not stand up well in a general application market. The major livelihood of this type of system results from a terminal addressing a large number of general applications, so that its product life can extend beyond a single level of application evolution. The identification of the general application markets and the development decisions will be discussed later in this paper.

Section B shows a general purpose terminal used in a specific purpose application. The typical environment contains one or more general terminals bound by special software, often with a dedicated minicomputer or microcomputer interfacing a computer system and the special purpose application. A heavy burden is placed on terminal support software to make the general terminals conform to the special requirements of the applications.

Section D shows the special purpose terminal satisfying a special application. The burden on application support usually resides in special hardware features or special microcoding. The trade-offs between Section B and D applications are the most easily understood. The basic problems are availability of components (including major OEM pieces), cost of hardware design, cost of software design, documentation, maintenance and training.

The user implementation aspects for these trade-offs and decision have been covered in numerous other papers and articles. The focus of this paper will be on the manufacturer rather than the user. Section B, C and D will be generalized as part of a larger problem of product planning in the next section, before bringing in examples of the decision making for applications.

THE BASIC INFLUENCES ON DEVELOPMENT OF A PRODUCT

The business units that form the product development group are:

- Administration
- Marketing
- Business Development
- Manufacturing
- Engineering (Hardware and Software)

The interaction of these units create the final product price and function, regardless of where the initial product idea

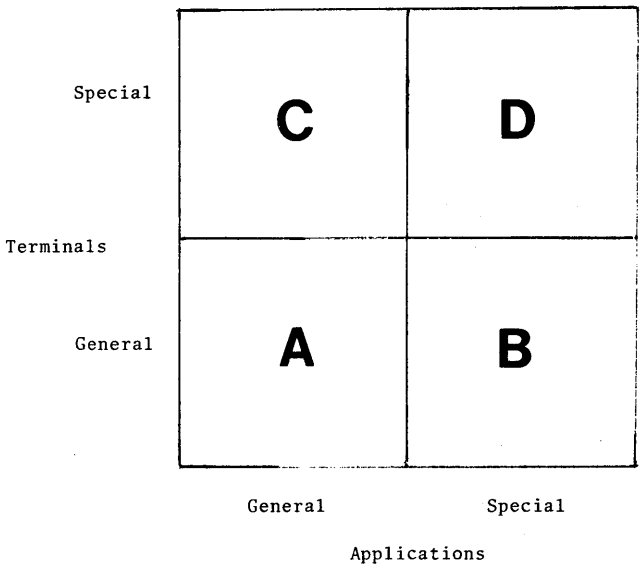


Figure 1—Sectional analysis of decision criteria

originated. Two basic outside influences complete the interactions:

- Customers/Market Place
- Competition

The interaction between all of these groups can be viewed as a feedback system using the approaches of Industrial Dynamics. Figure 3 shows a simplified feedback system for product function and Figure 4 shows a similar function for product price. Both price and function are inputs into each other's system interactions. In this simplified format the elements for Customer/Market Place and Competition are shown as constants but they have their own system dynamics which complicate this picture. For now these simple

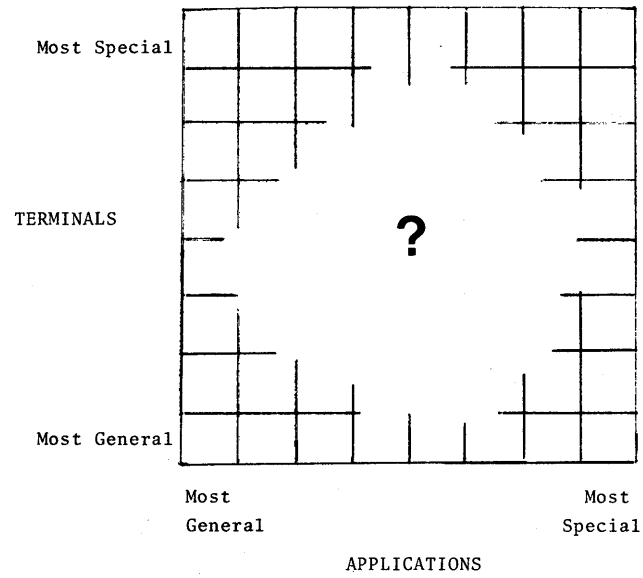


Figure 2—Sectional analysis for real world decisions

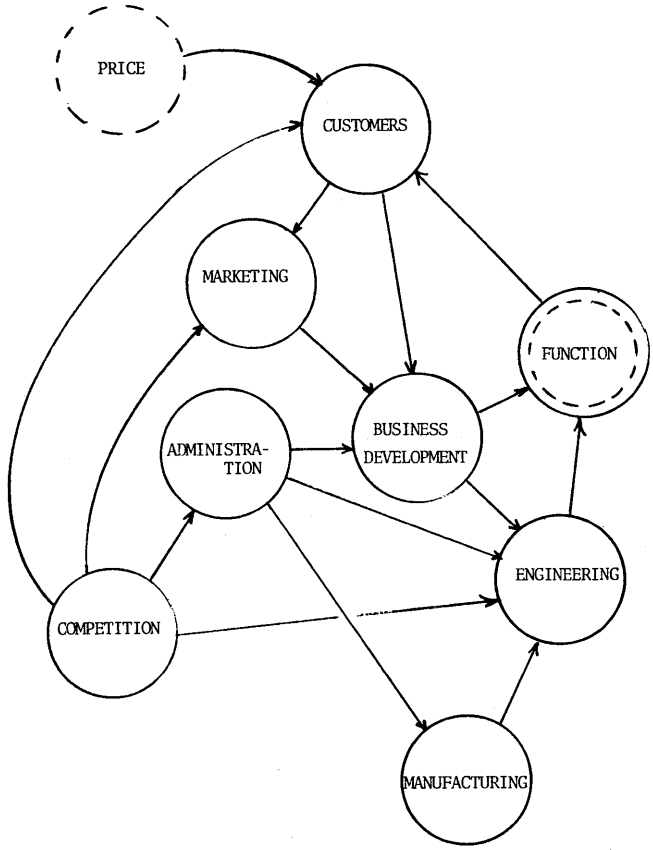


Figure 3—System dynamics interaction for function

techniques will be sufficient. The major intent is to simulate price and function determination in this feedback system until it stabilizes.

As each organization has its own management, department dominancies and interactions, there is no single system that solves how function and price will be determined. In the short run, the system described is likely to be astable. The non-linearities and administrative over-rides keep the real world stable. The major instabilities in the feedback model should be viewed as warnings, to help identify the failings and traps that occur in the process of product function and price identification.

The major problems of the product idea development are documented by many management oriented books and articles. Only a summary of problem areas will be covered in this paper.

- Dominance of a business unit—when all areas do not interact fully, an unbalance price/function decision may be made. Examples are:
  Manufacturing—An easily manufactured product may not have features required in the market place.
  Marketing—Features planned for good sales may cause poor system performance.
  Engineering—Design techniques may limit product utility or adaptability.
  Business Development—Long range goals may make the product initially unsalable.

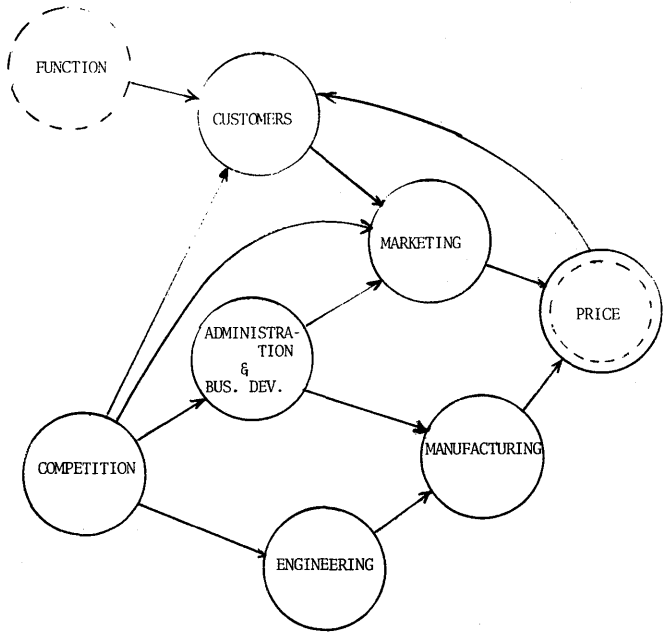


Figure 4—System dynamics interaction for price

- Inability to play "devil's advocate"—when a product idea is developed and all parties "jump on the band wagon," the balancing effects to round out a product may be missing.
- Communication problems may make all parties agree on the words used but have different concepts on the product usage, function and marketplace.

The "tales of woe" can run on indefinitely, but only a sample is necessary to realize that product planning is an art still rather than a science. The intention here is to skip this phase, and proceed to the business decisions in a terminal satisfying Sections B, C and D of the general purpose versus special purpose product.

PRODUCT TRADE-OFFS

For every user trying to decide which terminal to use and how to use it, there is a producer looking at that same problem. The producer has already finished his decision making, armed with the tools of market analysis and statistics. The producer also has his judgment and best guess working for him. When he produces a terminal product, the following factors lead to the general purpose versus special purpose trade-offs:

- Function
- Cost
- Performance
- Implementation
- Marketplace Size
- Competitiveness
- Profitability

These areas of decision and trade-off are interactive, in that a change in one will usually affect another.

An example of the product planning trade-offs can be shown for all these factors.

- *Function*—Function is the easiest factor to handle. The first user decision and equipment selection usually is "Does it do what I want?" Functions planned for a terminal product must be competitive to existing terminals already marketed. In a "plug for plug" replacement market, the unit must duplicate its competitor, and in a general market, the unit must exceed its competitor. This results in the terminal features escalation trends in the industry. A side note on function is that, if a good size market is buying equipment with "bells and whistles" it does not need but has no choice, reducing functions produces a special purpose terminal, generally at a reduced cost, and generally with a reduced market. Note the interaction with Marketplace Size, Cost and Competitiveness.
- *Cost*—The hardest factor to pin down is the cost tradeoffs. The second user decision on equipment selection usually is "Can I afford this?" If the equipment price is out of line with the competition in a marketplace segment, the equipment will quickly disappear from the market without some over-riding function or performance enhancement over the competition. Cost goals during product planning are made reflecting the product nature, split into two basic types: Price Cutters and Function Raisers. Price Cutters must always produce lower cost products, generally with identical functional features. Function Raisers must always produce more functional features (including performance), generally with a higher cost. Occasionally both can be combined and usually appear in the plug to plug replacement market. As cost goals are determined, the product is pushed into a special purpose or general purpose direction or preference. Interactions on cost are Function, Competitiveness and Marketplace.
- *Performance*—This is often viewed as a part of function, but is separate. A 300 Baud terminal and a 110 Baud terminal may have identical functions, but offer distinct performance differences. The tradeoff with special purpose and general purpose performance requirements are strongly coupled to price. The user wants all the performance he can get, if it does not cost him more money. Likewise, the user does not want to pay for any performance his application does not require. Both ends of the performance spectrum lead to a special purpose terminal, mainly because it attracts a limited marketplace that requires the additional performance, or desires the cost benefits. The interactions are Cost, Competitiveness and Marketplace.
- *Implementation*—Implementation is the most difficult portion of the trade-offs to judge. The user sees the packaging and styling and interacts with the operational characteristics. The external features are generally secondary to the equipment user. Internally, the product is strongly shaped by implementation. Functional and

performance growth are effected. Cost and profitability are determined by how the equipment can be manufactured and maintained. The producers emphasis on manufacturing or engineering can shape a product's long-term market and make it general or special purpose to achieve a market share and production volume required by the producer. A producer with low volume and R&D emphasis will tend toward specialized implementations and many product upgrades. A producer desiring high volume and low manufacturing costs will tend toward a generalized implementation to widen the marketplace. Implementation interacts with all the product trade-offs.

- *Marketplace Size*—The product direction to a wide or narrow market with a small or large volume is a producer's trade-off. The general purpose terminal usually is aimed for a large volume with a wide market. The special purpose terminals is usually aimed at a narrow market with a substantial volume. Some special purpose terminals are aimed over a wide market with low volumes in many narrow segments of the market. The decision on customer identification and subsequent marketplace size trade-offs should occur early in the product development cycle to allow the follow-up trade-offs on Function, Cost, Performance and Implementation. The marketplace determination is one of the most difficult ones, and the equipment user interacts in a secondary way, unless equipment is being manufactured to order for the user. The user interacts principally through surveys, sales representatives, and most important, by buying to his preferences in the equipment market.
- *Competitiveness*—This goes 'hand in hand' with marketplace size and forces the decisions on Function, Cost and Performance. The decision on how to compete and with whom to compete is part of all the previously discussed areas. This item also impacts all the above areas because timing of a product release and its subsequent distribution enter into the development cycle. The competitive factor pushes many trade-offs with a business decision to pursue markets, and also, how and when to do it. This factor in particular makes the simplified Figure 1 general versus specific market into the complicated Figure 2.
- *Profitability*—Last but not least important is the profitability trade-off. Profitability is almost always assumed, but is a trade-off along with competitiveness. Profitability is essential for a producer, but the degree and timing of profitability is a trade-off factor. The producer may not become profitable without limit, because the cost will not be competitive or the product will not be purchasable. The profitability in the early stages of a product may be a trade-off against long-term profitability. This factor affects all the other areas before manufacture and throughout the product life. As an unwritten goal it shapes many of the decisions producing special purpose or general purpose terminals.

The product trade-offs that occur in the development cycle affect the production of special versus general purpose equipment. The next step is the actual producers decisions which shape the product and its applications.

DECISION FACTORS IN PRODUCT PLANNING

The trade-offs, detailed above, lead into the actual product planning decisions. The decisions are the end result of the product trade-offs and are the (hopefully) formal documentation of the product direction. In order to concentrate on the basic decisions for product development and planning, a simple technique of 'problem busting' will be used. Problem busting is the subdivision of a major decision—in this case, product development—and generating simpler problems that may be attacked. In a sense, there will be a recombination of the areas discussed in the trade-off section, and responsibility will be assigned to the previously discussed business units.

The first major problem busting will uncover three basic problem areas for a product:

Existence Decisions
Salability Decisions
Administration Decisions

Each of these areas will be discussed and subdivided further to allow proper product analyzation.

The first area of existence decisions concerns fitting a planned product into the real world. The first subdivision is the existence of customer need. Does a customer base exist that needs the product? Does the customer base have the ability to buy the product? The existence of a ready to buy market produces products different from the products requiring an education before customer acceptance. The decision on existence of customer need is most influenced by Marketing and Business Development. The second subdivision is the existence or product viability. Can the product be implemented within the existing or developing technology? Can the product be manufactured within the existing capabilities? Will the product components be available in the future? The decision on the existence of product viability is most influenced by Manufacturing, Engineering and Business Development. The questions shown here break the problems down another level but will not be detailed here. The existence decisions should be made first in the product planning of equipment.

The second area of salability decisions concerns the determination of the product characteristics including the characteristics which make the equipment general purpose or special purpose. The first subdivision is performance and function salability. Can the unit be sold with the planned functions? Can it be sold with the planned performance? Does the unit still attract the customer base anticipated in the section on existence? These are most influenced by Engineering, Marketing and Business Development. This decision formalizes trade-offs on performance and function. The second subdivision is pricing and timing salability. Will the product sell to the customer base anticipated? Is the product on the market in time to sell or too early to be accepted? Can the product distribution handle the product?

These are most influenced by Manufacturing, Engineering and Marketing. This decision begins to lock the product into its role as special or general purpose. The third subdivision is salability against competition. The marketplace and the existing competition are analyzed against the planned product capabilities. Can the unit out-price the competition? Can the unit out-perform the competition? Can the unit 'steal' new business from the competition. Can the unit displace installed products of the competition? Will the unit displace the company's own installed products? The major influences on this are Marketing and Business Development. At this point the product characteristics are firm. The next major set of decisions cover the actual go ahead to produce a product.

The third area of administration decisions produces the final product with any long range modifications to the planned product. The first subdivision is administration of the risk and cost to produce. Can any component not be available in the future? Can any design problem not be solved within fixed constraints of the product? Can key individuals leave the project? Can manufacturing make the quantity forecast economical? Can any Assembly, Test and Quality Assurance areas invalidate basic design areas? This decision is influenced by Manufacturing, Engineering and Marketing. The second subdivision is administration of profitability. Can manufacturing cost and sales price produce a profit with known overhead? Can cost improvements be made? Can manufacturing improvements be made? Can component costs drop with volume and time? Can labor costs be handled? This decision is influenced by Engineering, Marketing and Manufacturing. The third subdivision is the administration of corporate requirements. Can this unit produce subassemblies in common with other equipment? Can this unit use subassemblies from other equipment? Can this unit solve an outstanding problem for another portion of the corporation? Can this unit produce side usage in other areas of corporate business with minor changes? This last area is influenced by all business areas including Administration. At this point the last of the product changes is made and the decision to go ahead with a product occurs. Function and detail specification and marketing literature follow in time, but now the product has become general or special purpose as a result of the previous trade-offs and decisions.

The intention of discussing decision criteria has stayed away from actual mention of terminals and remained as a general analysis. An example of the product planning in a terminal design will be shown in the following section.

EXAMPLE OF A SPECIAL AND GENERAL TERMINAL INSTALLATION DECISION

NCR Data Pathing, Inc. produces factory data collection equipment and systems as front ends for MIS oriented large computer systems. A special purpose data collection terminal, the 103, offered an employee badge reader, a punched card scanner, and keyboard input as well as 16 character message display and a backlighted message panel for output. This unit received wide usage in a narrow market.

In many installation sites, MIS capability was brought out to the factory floor with an IBM 3270 CRT display with keyboard, allowing a general inquiry system to the host computer system. These units often physically resided next to each other. The IBM 3270 is a very general purpose terminal and was used in conjunction with a special purpose terminal in a fairly clumsy manner. As a large number of these combinations showed up, a special purpose terminal with a degree of generality was planned to replace the combination.

DPI introduced the MIT-133 as an IBM 3270 replacement specifically for the factory data collection environment. The MIT-133 had the IBM 3270 CRT and keyboard, but also added a special keyboard for factory environment with large keys and simplified sequential positioning. Also the badge reader and card scanner were added. The keyboard was placed at an angle to make operation easier from a standing position which is typical in factory inquiry, and a quadruple size character font was added to improve readability of the typical short factory messages. This special size character set replaced the 16 character message display, and the backlighted message panel. Special software in the DPI front-end formats the CRT screen with the same data editing capabilities available on the DPI 103.

The unit is a special purpose terminal in a special purpose application displacing a general purpose terminal in a special purpose application. The unit is a function raiser not a price cutter, and the MIT 133 is actually higher priced than the IBM 3270. This unit has extended functions that get the factory data collection job done more efficiently, and as a result, save considerable cost from the main job of MIS, saving manufacturing money.

The decision process covered the customer need and product viability, demonstrated by the actual co-existence of the DPI 103 and the IBM 3270. Performance matched the existing units and was improved by creating a single data entry terminal unit. Pricing and timing recognized additional capability for additional cost when there was a performance benefit. Competitive capabilities were satisfied by incorporating all IBM 3270 features as well as adding new capabilities. Administrative decisions cannot be detailed.

The decision shown here is only a Section B versus Section D, but the decision criteria are valid across all product planning areas.

REFERENCES

1. Forrest, J., *Industrial Dynamics,* MIT Press, MA.
2. Schrello, D. M., "Be It Ever So Humble, There's No Substitute for Product Planning," *EDN,* January 5, 1976.
3. Grossman, L., *The Change Agent,* AMACOM, NY.
4. Lippett, R., *The Dynamics of Planned Change,* Harcourt, Brace, NY.

Making a special purpose terminal work in a general purpose systems environment

by DONALD J. BIRMINGHAM
NCR Data Pathing Inc.
Sunnyvale, California

THE FACTORY DATA SYSTEM

Factory Data Systems have many application requirements not normally satisfied by standard general purpose terminals. In the past twenty years there has been an evolutionary growth of job oriented terminals to satisfy the unique requirements of the factory. To maintain the pace of the current expanding system requirements, the special purpose terminals must communicate on a transaction basis with a general purpose data processing system.

A schematic of a comprehensive factory data system is shown in Figure 1. The factory terminal system has many requirements which include both general purpose and special purpose application parameters in the following areas:

- Time and Attendance
- Labor Distribution
- Job Tracking
- Material Control
  - —Raw Material
  - —Work in Progress
  - —Finished Goods
  - —Shipping and Receiving
- Inquiry
- Tool Tracking
- QA Reporting

Most of the above terminal applications must communicate directly with a large scale general purpose data processing system. This paper outlines the system implementation that Data Pathing Inc. uses to connect an interactive factory data collection system on line to a large scale IBM data processing system. DPI offers special purpose job oriented terminals, general purpose terminals, and terminals which alternately operate in a special purpose job oriented mode and a general purpose mode.

THE SYSTEM FLOW

Figure 2 presents the desired flow of transactions from the job oriented factory terminals through a DPI system control processor to a "host" general purpose data processing system. Not all transaction types have the same real time processing requirements with the host data processing system. Some of the different system requirements include:

- Immediate and Mandatory Interaction
- Queuing for Load Leveling
- Queuing for Time Dependent Processing
- Processing Off-Line for Faster Response
- Processing Off-Line for Back Up Purposes

In most systems the main plant data base is located on the host processor. A prime example of this in the large IBM environment is an IMS data base oriented system.

TYPICAL FACTORY JOB ORIENTED TERMINAL

A typical job oriented special purpose factory data collection terminal is detailed in Figure 3. This particular terminal is designed for "community" applications. A community of workers runs random or time dependent application transactions on the terminal. Examples of the transactions are as follows:

Time Dependent	—Time & Attendance (In & Out)
	—Last Job Checkout (Shift Change)
Random	—Material Move (Function Complete)
	—Material Receipt (Material Available)
	—Labor Transaction (Job Complete)

The DPI SDT 107 is a multi-function terminal which can run the transaction applications gamut from time and attendance through material move to limited inquiry. To do these transactions the terminal incorporates the following features.

- Transaction Descriptions (Applications Menu)
- Transaction Select Keys
- Operation Instruction Mask
- Function Indications
- Keyboard (Numeric and Alpha)
- Alphanumeric In Line Field Display
- Function Keys

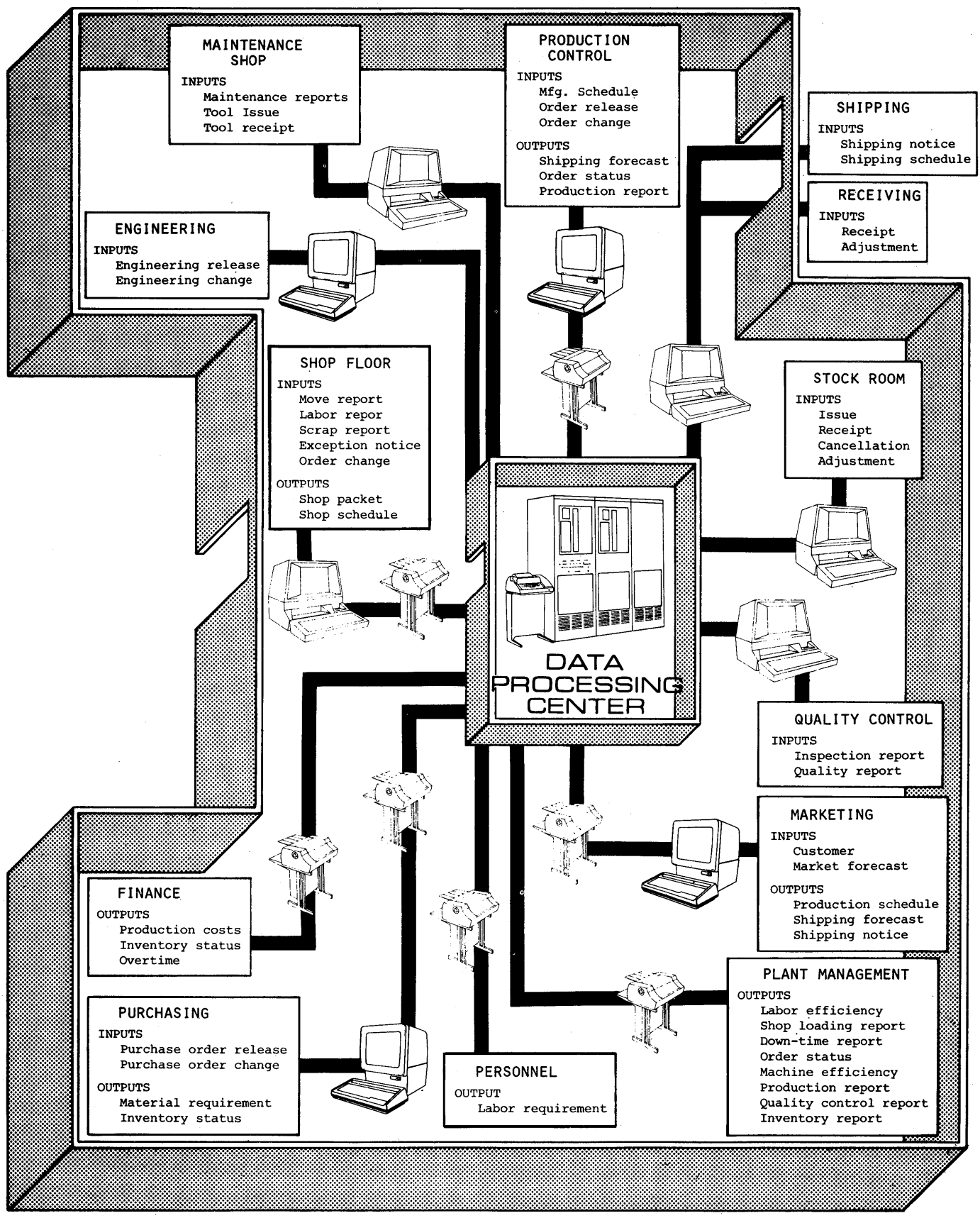


Figure 1—A comprehensive factory data system

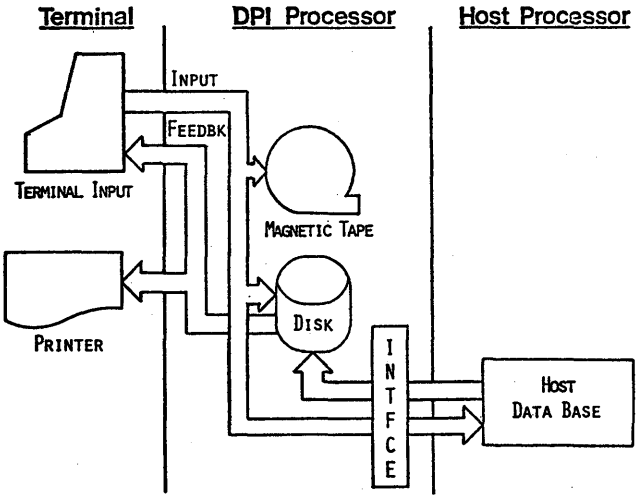


Figure 2—Data disposition

- Badge Reader
- 80 Column Punched Card Reader
- 22 Column Punched Card Reader

THE MULTI FUNCTION FACTORY TERMINAL

The growth of different application terminal requirements has led to the side by side placement of both job oriented and general purpose terminals each installed for specific applications. As new applications are added it becomes a problem as to which terminal the transactions are assigned as most applications have both general purpose and special purpose attributes. To solve this dilemma DPI developed the MIT 133 Factory terminal. This terminal can alternately operate as either a special purpose job oriented terminal or as a general purpose keyboard CRT terminal. In the general purpose mode it operates as an equivalent IBM 3277 terminal. As shown in Figure 4, it has both the main attributes of the general purpose and the job oriented special purpose terminal.

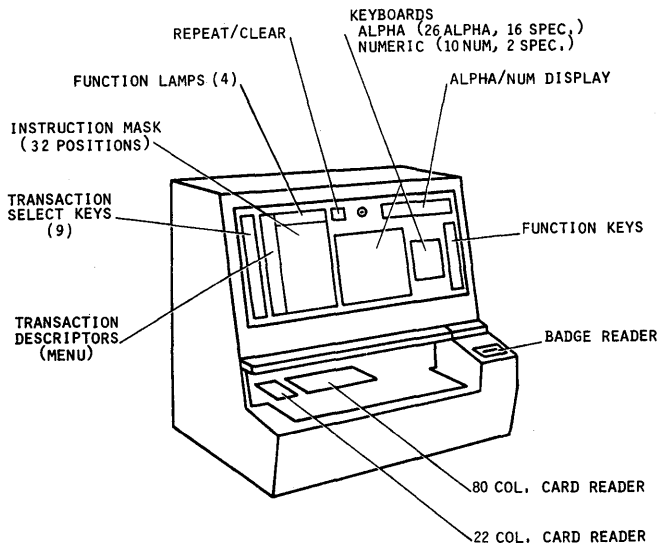


Figure 3—DPI 107 multi-function terminal

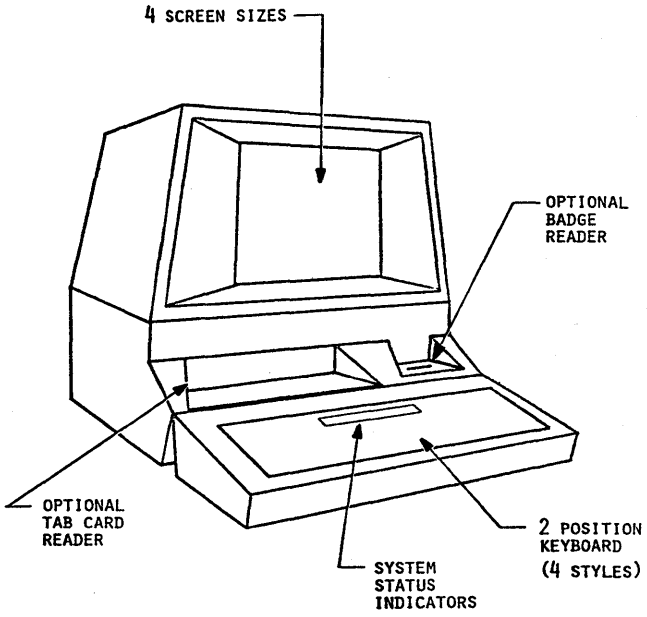


Figure 4—MIT 133 display station

General Purpose		*Special Purpose*
15 inch CRT Screen	w	four different screen sizes
Alphanumeric keyboard	w	special job oriented layouts
Basic system indicators	w	Expanded System Indicators
		Optional Badge Reader
		Optional Tab Card Reader

Figure 5 presents a point by point feature comparison between the DPI MIT 133 and the IBM 3277. Most of the added features were specifically added to allow for alternate or mixed operation in the general purpose or special purpose job oriented mode.

INTERFACE CONTROL SYSTEM

The DPI 3270 Interface Control System (ICS) is an optional system module to interface a DPI system with any host processor system that will support the IBM 3270 terminal system. The module is independent of the specific access method used in the host computer system. The applications can be programmed for interaction with IMS, CICS, TCAM, TOS or any other access method which supports the IBM 3270 general purpose CRT terminal system.

Within this capability the DPI system has a full range of unique interface arrangements to enhance the power and flexibility of the terminal applications.

ICS is an emulation of the IBM 3270 information display

FEATURES	DPI MIT 130	DEVICE BEING EMULATED
LIGHT PEN		✓
AUDIBLE ALARM	✓	✓
KEY LOCK	NOT REQUIRED	✓
MAGNETIC CARD READER		✓
SYSTEM INDICATORS	13	4
CHARACTER SIZES	2	1
PRINTER SPEED	120 CPS	66 CPS
PRINTER CABLE	2 WIRE	COAXIAL
IN-PLANT LINE USE	2 WIRE	MODEMS
EMPLOYEE BADGE READERS	✓	
CARD READERS	✓	
NON-CLERICAL KEYBOARDS	✓	
BLINK FIELDS	✓	
STANDING OPERATION	✓	
PLANT TIME SYNCH	✓	
DUAL TERMINATION	✓	
DATA COLLECTION	✓	
HOSTILE ENVIRONMENT DESIGN	✓	

Figure 5—MIT features comparison

system. Two types of system configurations are supported within ICS:

Figure 6: Direct IBM channel attachment referred to a local attachment (Emulation of IBM 3272)
Figure 7: Communications line attachment hereafter referred to as remote attachment (Emulation of IBM 3271)

The figures show the "Virtual System" which the host IBM processor system "thinks" it is controlling and the "Real System" which is being implemented by DPI.

SPECIFIC TERMINAL OPERATION UNDER ICS

Under ICS DPI terminal can operate in either the "transparent" or "data collection" mode.

- Transparent Mode is a one for one operation of a DPI

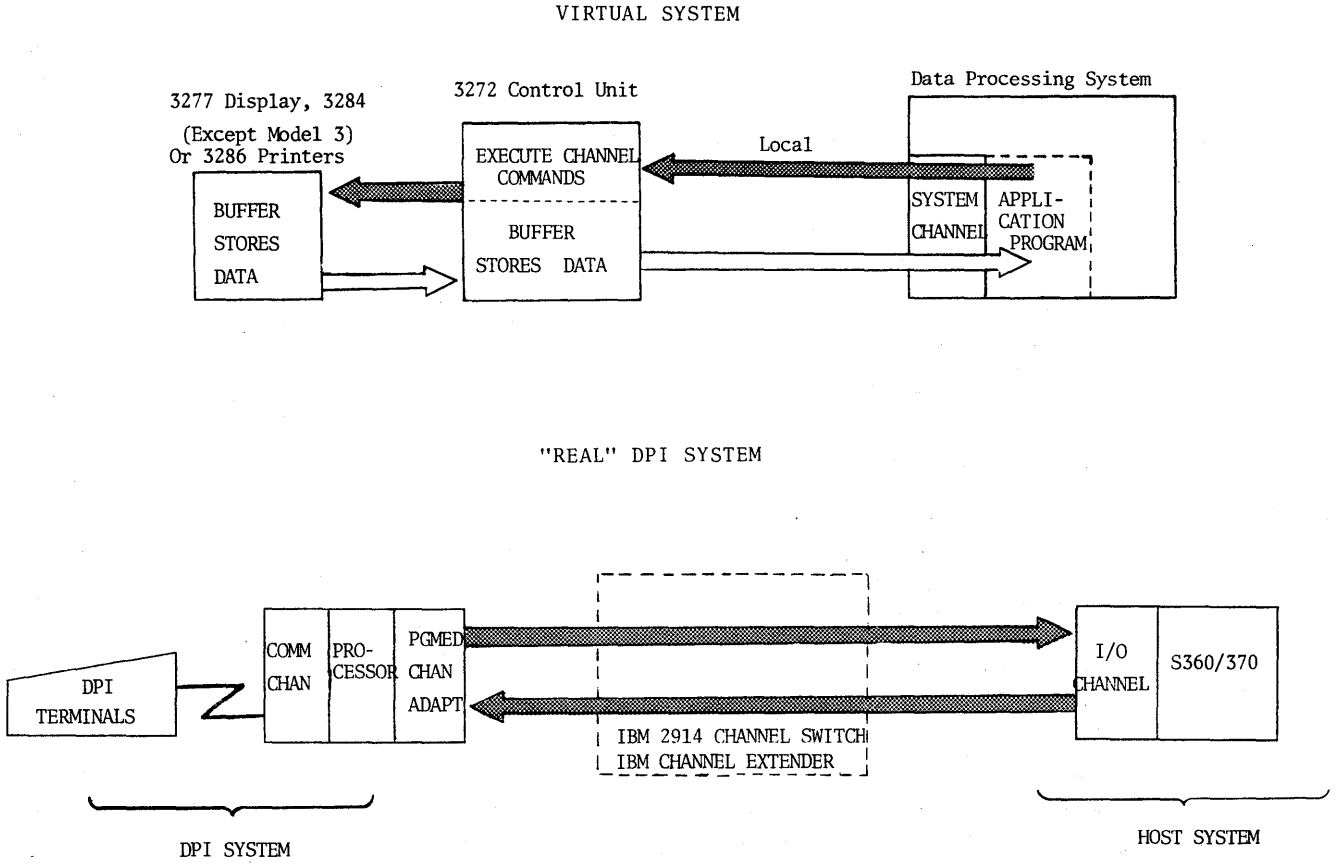


Figure 6—3272 local attachment

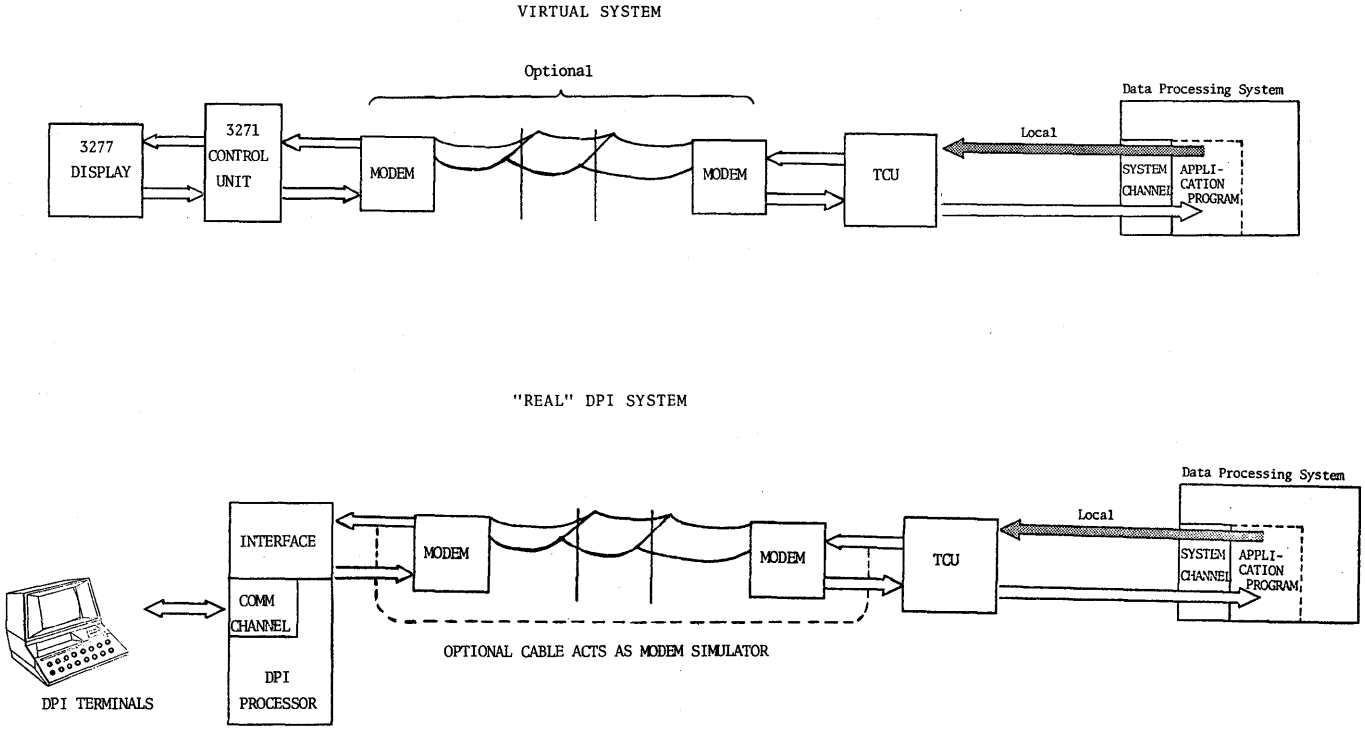


Figure 7—3271 remote attachment

terminal as an equivalent IBM 3277 terminal. The screen format control resides in the host processor and all transaction processing from an applications standpoint is executed in the host processor.

- Data Collection Mode can be either a one for one or group operation of DPI terminals for IBM 3277 terminals. The screen format control resides in the DPI processor. Final transaction processing can be executed in either the DPI processor or the IBM host processor.

A terminal such as the DPI SDT 107 is normally applied only under the data collection mode of operation. As an example it is not uncommon for each of two hundred DPI 107 terminals to appear to an IBM system as the physical equivalent of two IBM 3277 terminals (one for input and one for output). Limited job function terminals such as a dedicated time and attendance terminals are also included under the same emulation mode. The logical terminal addressing is accomplished in the host processors applications.

The DPI MIT 133 terminal can be applied either in the "transparent" or "data collection" mode of operation. It is not unusual for the MIT 133 terminal to be applied in the mixed mode where specific transactions are transparent and others are run in the data collection mode. In this type of application a specific terminal may appear to the host processor as two different logical terminals (one transparent and the other a data collection mode terminal). In such a case the transparent mode of the multi-mode terminal mode would appear to the host as a one for one physical terminal. The data collection mode of the same terminal can appear as either a separate one-for-one physical terminal or grouped as an individual logical terminal with other data collection mode terminals under a single physical virtual terminal.

A data collection mode application can be processed against both a reference (or fast reaction) data base in the DPI processor, and the main data base in the host processor. A "work in process" (WIP) file is the typical factory environment example of the type of data base located in the DPI processor.

SUMMARY

- The concept of having a special purpose terminal system appear to a host processor as a general purpose terminal subsystem has provided a most flexible system method of interfacing factory job oriented terminals to general purpose data processing systems.
- The concept coupled with special purpose terminals such as the DPI MIT 133 designed to operate in both the general purpose and special purpose job oriented mode provides for multi-operation of software control executed from either the best system or the DPI processor.
- The IBM 3270 is an ideal emulation target for providing multi-function on-line operation of job oriented terminals.
- Potential users need not worry about obsoleting existing working 3270 CRT software in the host processor when adding or evolving to a more sophisticated interactive on-line factory terminal system.
- The implementation of virtual logical terminals frees the software writer on the host from redefining all communications devices as new applications and terminals are added or modified.

A panel session—Future developments of special purpose terminals

SESSION CHAIRMAN—REIN TURN

TRW Defense and Space Systems Group
Redondo Beach, California

Panel Members

Carolyn Dunning—PCC Business Systems
Santa Ana, California
M. B. Herscher—Threshold Technology Inc.
Delran, New Jersey
Robert H. Anderson—The Rand Corporation
Santa Monica, California

INTRODUCTION

Many types of special purpose terminals are being developed for various applications to increase the performance and cost-effectiveness of man-computer interfaces. As discussed by the speaker and panelists in this session, future advances in computer technology will provide new design options, such as providing built-in intelligence through the use of microprocessors and memory chips. Thus, in future terminals it will be possible to do much more processing and store much more information than at present, and it will be possible to incorporate more effective security features and use more natural man-computer communication modes.

Donald Kovar in his paper "The Future of Special Purpose Terminals" (in this volume) examines current trends in computer terminal development and extrapolates these into 1978-1988 time period, with special emphasis on the transition from large scale integration (LSI) to very large scale integration (VLSI). He offers projections concerning the demands for special purpose terminals and on the ability of the industry to respond to these demands. His explorations are examined in more detail by the panelists who have chosen to concentrate on specific technology areas as well as on the extent in which future advances in technology are likely to affect terminal systems presently in use.

THE TERMINAL OF THE FUTURE IS A MINICOMPUTER

Panelist Robert H. Anderson of The Rand Corporation, Santa Monica, California asserts that the single most important attribute of the interactive computer terminal of the future is that it contain a general-purpose computer with the approximate characteristics of a present-day minicomputer. Representative of the set of services to be provided by the terminal of the future is a compatible package of user services developed at Rand on a PDP 11/70 minicomputer under the UNIX operating system.[1] The major systems within this software package (in addition to the standard UNIX services) are: a CRT-based two-dimensional text editor (NED);[2] an electronic message system (MS);[3] a reminder facility (REMIND) capable of sending a message or initiating a program at an arbitrary date/time; the RITA system[4-6] for creating user agents defined by sets of condition-action rules; an extension of the operating system called "virtual terminal UNIX" capable of providing the user windows onto multiple simultaneous processes; and an interface to an electronic data network (in this case, ARPANET). The described systems provide a coherent, symbiotic set of services that should co-exist. They should be provided on computing power local to the user for several reasons: their availability is not subject to external schedules and control; access to these services does not require lengthy login and authentication protocols to time-shared systems or data networks; data privacy can be maintained without reliance on central computer system security or transmission lines; programs within the user's terminal can act as "agents" capable of interacting automatically with external systems on behalf of the user, thus providing tailored interfaces and freeing the user from mundane, routine interactions. He concludes that the most valuable service a user terminal can provide is general-purpose computing, both for stand-alone use and as an aid in accessing remote information systems.

THE FUTURE OF SPEECH INPUT TERMINALS

Another panelist, M. B. Herscher of Threshold Technology Inc., Delran, New Jersey, points out that voice data entry, long recognized as an ultimate step towards simplifying communications between a human and a machine, is now a reality. Over 200 voice terminals currently are in operation in a variety of industrial and commercial applications in eight countries around the world. New speech input terminals using microcomputers have been introduced

which can replace and/or complement intelligent CRT/keyboard stations by enabling the user to enter data by voice. In many applications where hands and/or eyes are occupied in normal work environments, voice input terminals permit source data to be captured in an extremely efficient manner. Since these terminals normally provide input verification and editing capabilities, virtually error-free data can be captured. Speech input terminals also are being used for programming NC machines, aids to the handicapped, various material handling applications, and in a variety of training applications.

Currently, almost all of these speech input terminals recognize isolated words or phrases from a limited vocabulary subset and require that the user preprogram (train) the system for his or her pronunciation of the vocabulary. These recognition systems usually require bandwidths of up to seven KHz although some of these terminals can operate over phone lines. A complementary application of current speech input terminals is to recognize (verify identity) of individual talkers from their speech characteristics. The first secure access system for a computer room using speaker authentication has been in operation for several years.

The potential applications of voice data entry are determined by the economic advantages of voice input as compared to other alternative data entry devices and/or techniques. As the cost of voice data entry terminals decreases, more justifiable applications will arise, particularly when the true costs of data capture are considered. In some cases, voice input offers advantages which far outweigh the direct labor savings and permit certain operations to be achieved which could not easily be accomplished using alternative data input techniques.

Almost any system using intelligent CRT/keyboard terminals for data input could use speech input terminals. Virtually any factor or office system which is now directed by instructions from either a keyboard or a computer could be controlled by verbal commands. With current speech technology, the human voice simply replaces the mechanical keyboard or computer control, often with distinct operational and economic advantages.

Mr. Herscher expects the progress in the future for speech input terminals to take several directions. Larger size vocabularies will become practical for isolated word recognition systems. Some systems will be developed which will not require the user to pre-train the system vocabulary, although higher recognition accuracies always will be achievable for the adaptive system. Reliable operation over switched phone lines will open up new applications of speech terminals. Continuous speech recognition terminals will be available within the next one to five years—initially, connected digit recognition, followed by larger vocabularies. Combined recognition/speaker verification terminals will be available for accessing computer data bases and for electronic funds transfer, eventually over the phone lines.

He concludes with the observation that additional progress will be needed to achieve practical continuous speech recognition. Difficulty will mount in a geometric progression as more sophisticated speech systems are attempted. Once the cost barriers are overcome, the benefits will mount quickly, so the results promise to be worth the effort and investment. Over a ten year period, the first terminals for recognizing relatively natural speech input will begin to appear.

KEYBOARD DATA ENTRY ISN'T DEAD YET

The third panelist at this session, Carolyn Dunning of PCC Business Systems, Santa Ana, California addresses the impacts of technology advances on current data entry terminals and their use. How long does it take for an incumbent industry or profession to yield to the onslaught of new advances? She observes that keyboard data entry is supposed to have been 'dying' for over 20 years, ever since the introduction of optical character recognition (OCR). Each time a new device is developed the cry "keyboard data entry will become obsolete in five years" is heard again. Yet through all of this, there are more data entry keyboards of some type in use today, and more are projected for the future.

Special purpose terminals such as point of sale (POS) and new means for data entry such as voice recognition, may indeed have an impact on traditional data entry. But it is unlikely that it will be very great. Special purpose terminals are too specialized, and so is voice data entry (which is also too slow for high volume processing).

Distributed processing and distributed data entry may have a big impact on the traditional centralized data entry concept. However, most distributed data entry still uses a keyboard of some type. While it may look more like a typewriter, it will still use a keyboard. The emphasis may not be on quantity, or production, but on timeliness and accuracy which will be increased by having the person most familiar with the data entering it.

According to forecasts prepared by International Data Corporation (IDC), machine oriented systems such as OCR will experience approximately a 5 percent growth between 1976 and 1981. Keyboard type data entry will see an increase of nearly 100 percent, and this takes into account an estimated 16 percent decrease of centralized data entry systems, such as key punch and key-to-disk.

The predicted growth will come from increased installations of distributed systems, both single station and clustered as well as increased use of conversational and editing terminals. All these employ keyboard devices for data entry.

Another indication that keyboard data entry is still very much alive, even in a centralized environment, is the shortage of data entry operators. The newspapers are full of ads for these people. In addition, many companies are offering a reward for anyone who sends a qualified applicant for data entry. Other companies are taking more trainees or training people themselves. When operators can be found, they cost more. In the decade from 1966 to 1976, the average salaries for key entry operators in all grades increased by more than 50 percent.

While certain types of data can be entered in a 'dumb' sense, exactly as seen, the majority will still take some device capable of thinking and making decisions. The best device for this is still the human brain. Working in conjunc-

tion with the eyes, and fingers, it is the most accurate, rapid and practical method of entering data available, not just now, but for many years to come.

REFERENCES

1. Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM,* Vol. 17, No. 7, July 1974, pp. 365-375.
2. Bilofsky, Walter, *The CRT Text Editor NED—Introduction and Reference Manual,* R-2176-ARPA, The Rand Corporation, Santa Monica, California, December 1977.
3. Crocker, David H., *Framework and Functions of the 'MS' Personal Message System,* R-2134-ARPA, The Rand Corporation, Santa Monica, California, December 1977.
4. Anderson, R. H. and J. J. Gillogly, *Rand Intelligent Terminal Agent (RITA): Design Philosophy,* R-1809-ARPA, The Rand Corporation, Santa Monica, California, February 1976.
5. Anderson, R. H. and J. J. Gillogly. "The Rand Intelligent Terminal Agent (RITA) as a Network Access Aid," *AFIPS Conference Proceedings,* Vol. 45, AFIPS Press, Montvale, New Jersey, 1976 NCC, 1976, pp. 501-509.
6. Anderson, R. H. et al., *RITA Reference Manual,* R-1808-ARPA, The Rand Corporation, Santa Monica, California, September 1977.

The future of special purpose terminals

by D. G. KOVAR
TRW Communications Systems and Services
Torrance, California

INTRODUCTION

In his book, *Future Shock,* Alvin Toffler dramatized the fact that we live in a rapidly changing world, in which most of us must constantly struggle to merely stay current. With this thought in mind, attempts at predicting the future must always be approached with some trepidation and perhaps a dash of humility. Nevertheless, it is the intent of this article to postulate the future of Special Purpose Terminals over the period of the next 10 years (1978-1988). This will be done by assessing the present state of this field, examining current trends, and extrapolating these findings to produce a range of future expectations. Although the year 1984 falls within the period of study, it is the opinion of the author that "Big Brother" will *not* be watching you on a Special Purpose Terminal during this time.

For the purposes of this paper, we shall define the term Special Purpose Terminal to mean computer input/output devices which are *used by people* and which have physical and/or functional characteristics that are specialized for a particular application or environment. For example, using this definition, we would conclude that a laser scanning, supermarket point of sale unit integrated with the check stand[1] is a Special Purpose Terminal, but on the contrary, a Teletype is not. There are, of course, many examples in which the line between special and general purpose is less clear than the above. Consequently, the definition given is useful as a guideline, but not as a rigid rule. Using this definition guideline, we find the following application areas to be currently active in the field of Special Purpose Terminals:[2]

Point-of-Sale
- Retail Department and Specialty Stores
- Drug and Discount Stores
- Supermarket
- Service Station

Banking
- Teller Assist
- Automated Tellers
- EFT

Industrial/Medical
- Factory Data Collection
- Hospital
- Security Access Control
- Rapid Transit
- Parking Lot

Travel and Entertainment
- Airline Ticketing and Reservation
- Rental Car
- Entertainment Ticketing
- Automated Traveler's Check Dispensers

Miscellaneous
- Library
- Law Enforcement

As time goes on, this already impressive list will be strengthened and some potentially very large new categories will be added.[3] These are as follows:

- Office Automation
- Electronic Mail
- Conferencing
- Distributed Work Environment
- Consumer Applications

The requirement for existence of any of these terminal devices (both current and future) is primarily a question of economics. Advances in technology have made it possible to produce increasingly sophisticated terminals at improving costs and reliability. Yet, in order for equipment to be commercially produced, there must be an adequate demand for the functions provided. Specifically, there must be a large enough separation between the economic value (real or perceived) and the cost of production in order to motivate investment by *both* the supplier and the buyer. This can be visualized as shown in Figure 1.

The relationships of Figure 1 must hold. Otherwise, the product will not be produced even if it is desired, or it will not be sold even if it is produced, or it will be produced and sold for a time period until either buyer or supplier with-

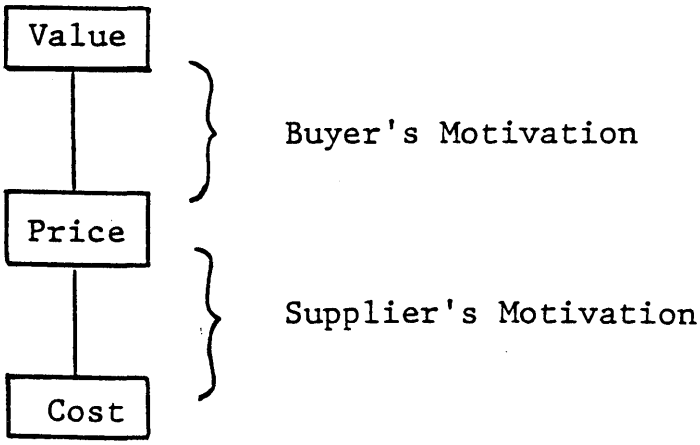


Figure 1—Product viability

drawal occurs. Consequently, it is insufficient to examine the future by studying only the technology. One must examine both sides of the equation and attempt to project product instances where satisfactory value/cost relationships will be likely. This is the methodology used in this paper.

TECHNOLOGY

Dramatic advances in technology have made the modern Special Purpose Terminal a very powerful subsystem. Many of the terminals of today already use one or more microprocessors[4] to control the logical operation and have several 10's of K Bytes of programmable storage. Having this compute power in the terminal allows for such functions as operator prompting, input data validation, transaction-oriented arithmetic, interactive communication with a higher-level computer, and complete stand-alone operation if necessitated by communication failure. This compute power has been made available primarily by advances in the state of the LSI art.[5] It is expected that such advances will continue in this area leading to the realm of VLSI.[6] This will in turn make it practical to house substantially more compute power in the terminal than is possible today. However, before commenting on the significance of this, let us first take a slightly broader look at the technological needs of the Special Purpose Terminal.

The Special Purpose Terminal is primarily a person-computer input/output device. The basic methods used to perform the input/output functions are shown in Figure 2. As can be seen, the variety of input/output means is very large and each of these means involves its own peculiar technology. It should be further noted that only a few of these means directly benefit from advances in LSI technology. Consequently, so far during the 70's, we have experienced considerable disparity between the *rate of change* of technology that is used for the internal logic vis-a-vis that which is used for the input/output functions.[7,8]

The result has been a Special Purpose Terminal price versus time curve that has been almost flat and, in some cases, has even risen slightly. At the same time however, logical capability (or performance, if you will) has risen in direct proportion to LSI progress, thereby producing significant gains in the performance/price ratio. So, one could characterize the experience to date as being essentially flat on price with rising performance rather than fixed in performance at declining prices. In order to be economically viable, such a situation must be accompanied by a rising demand for performance coupled with a tolerance for constant price. So far, this has been the case.

Whether or not we can extrapolate from this constant price, rising performance, profile is a key question in projecting the future. The rising performance part seems a safe bet, so let's examine that first. Depending upon the geometric regularity, current production level MOS LSI could be described as having reached the 10-20 thousand elements/chip level of density. Advances in mask making technology such as electron beam methods promise to raise that density to the one million-plus level during the period of study.[9] In anticipation of these improvements, the term VLSI is coming into popular use. This suggests that we can expect 32-bit computers, one million-bit memories, and combinations of these on individual chips.[10] Perhaps this will introduce the cliche of "the mainframe on a chip."

How will we spend that compute power? What good is that much horsepower in an individual terminal? The current economic picture for the Special Purpose Terminal would suggest that the new levels of compute power be spent on attacking the input/output problem—both for the operator *and* and the programmer. Increased compute power can improve the economics of some of the input/output devices by absorbing more and more of the special electronic processing that is externally contained in current equipment—i.e., laser scanner, OCR wand, etc. Increased compute power can be used to make new input/output methods possible—i.e., fingerprint, signature, voice, etc. The programmability of the Special Purpose Terminal is a very important feature in providing the adaptability and flexibility needed to meet changing requirements. The value of this feature should be expected to increase as time goes on. Consequently, "wasting" some of these new levels of compute power and memory space on higher level, more person-like (less machine-like) programming languages and tools can be anticipated.[11] So, we should look forward to Special Purpose Terminals with some increased function and with substantially improved person-machine interfaces.

In addition to the improvements expected in the basic logic function which are spurred on by advances in LSI, we can also see some considerable advances in the local storage function. Local storage is used primarily to collect data generated by the terminal for subsequent transmission and to store programs used by the terminal. Normally such storage must be non-volatile, and this requirement is currently satisfied by using small disc or tape units. The magnetic bubble memory[12,13] is an attractive alternative technology. Just now being commercially introduced, the bubble memory is expected to surpass these electromechanical devices

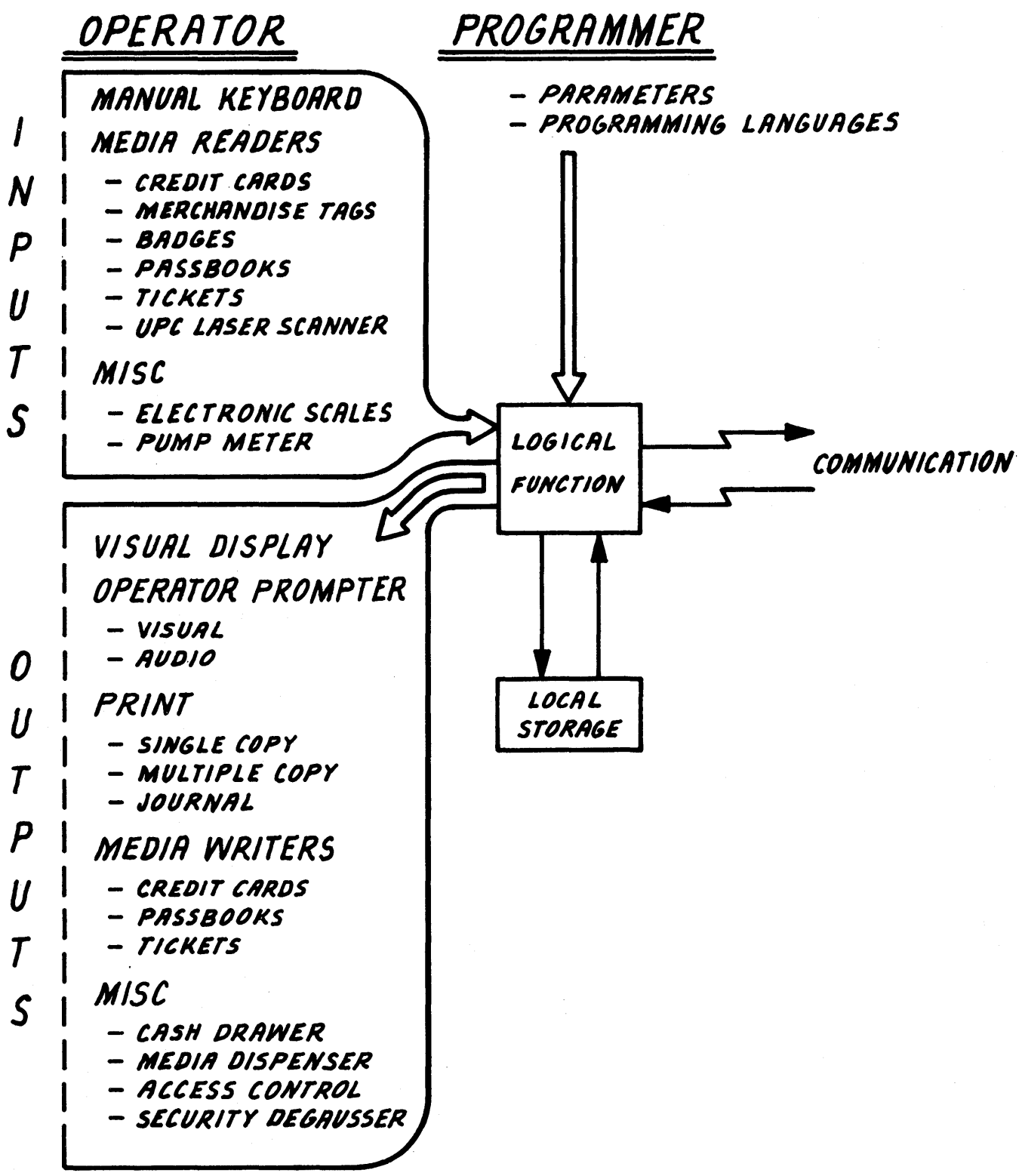


Figure 2—Special purpose terminal interfaces

in performance/price ratio's by 1980. High expectations for continuing bubble memory density improvement beyond 1980 are realistic because such improvements are dependent upon the same improvements in mask making technology as is the case for LSI. Therefore, bubble memory is expected to be used as direct replacement for current local storage systems. In addition, the constantly improving bubble memory economics (ala LSI → VLSI) will make it possible to deploy non-volatile local storage in terminal environments for which such storage is not currently cost-effective.

As previously mentioned, while dramatic advances in some portions of the Special Purpose Terminal technology can be expected, there are some items that resist. Among the technologies implied by Figure 2 displays, printers, packaging and power supplies stand out as the most unyielding. While many new technologies have been introduced for small terminal displays such as liquid crystal, plasma, and LED, they are still not economic for displays of several hundred characters.[14] In fact, compared to CRT, the disparity can approach an order of magnitude.

Because of this difference, and because of the pressure (for improved person-machine interfaces) which will emphasize larger displays (including graphics), the CRT will continue as a major display technology. This projection is made by assuming that a "breakthrough" based upon some currently unknown technology does not take place. Although several printer technologies are being pursued,[15] i.e., impact dot-matrix, thermal, electro-static, ink-jet,[16] etc., they all have moving mechanical parts and suffer from relatively high initial cost and continuing reliability problems. The printer has sometimes been described as the "Achilles' Heel" of the Special Purpose Terminal. However, some cost improvement will be realized by minimizing or completely avoiding the printing requirement through the use of solid state technology as an alternative method for information processing. For example, local storage by bubble memory can eliminate the journal and information distribution by electronic means can offset the need for printing on multi-part forms. The printer will not go away, however, as there will be a continuing need for some hard copy for an indefinite future period. In the packaging and power supply technologies we can expect some improvement via superior materials, components, and design architecture but we should not count on these advances producing any more than a hedge against inflation. As a result of the above observations, the constant price phenomenon is expected to continue for some time except for those gains that can be realized by substituting solid state for printed information handling.

The picture being painted is one of constantly improving performance and functionality under a fairly constant or, at best, modestly declining price umbrella. We can expect those 1988 Special Purpose Terminals to be programmable in very high level languages, to provide solid state, non-volatile local storage, to offer highly economic input media readers, and to introduce some elementary versions of alternate human input such as voice, fingerprint, etc. But where will this technology be used? What actual products can be expected? Let us turn to that now.

DEMAND

Today we find Special Purpose Terminals well entrenched in the major retail department stores and financial institutions.[17,18] The demand for these terminals has come from the values derived from the improved automated assist functions for clerks and tellers and from the improved velocity and accuracy of information associated with consumer transactions in the institutions. Such values can be expressed in terms of improved productivity, asset management, and customer service. In a society confronted with a growing rate of change, these values can be expected to continuously increase. Therefore, we project a continuing demand for Special Purpose Terminals by the retail and financial institutions. Here, we will see heavy emphasis on improving the programmability to keep up with the rate of change, and in moving more and more to automated input means as the economics improve. Some customer fingerprint optical processing for identification may come into use but demand for widespread use of voice input/output is doubtful in the active lobby-like atmosphere of these environments.

The question of demand for Special Purpose Terminals for use in consumer transaction electronic funds transfer (EFT) is fascinating in its complexity.[19] Here is a case in which the necessary technology already exists. But who is the terminal buyer? Who benefits? Who runs the multi-bank, multi-retailer interconnect network? Can terminals be legally shared? What about banking legislation in general? What about consumer protection legislation? Satisfactory answers to these questions will be needed before significant investments are made in EFT.[20]

In spite of these problems, there are some strong motivators for EFT. The banking institutions face the same rising costs of real estate, and energy as are pervasive in society in general. In addition, the growing volume of paper in the system is a constant threat to processing saturation. On a more positive note, EFT should be able to deliver a more expeditious and broader range of financial services to consumers.

So, how will this tug-of-war be decided? It is projected here that consumer transaction EFT will come about but it will be a slow process. Many experiments by individual banking institutions have been conducted and more will follow.[22] The anticipated progress toward true distributed financial institution EFT is projected to still be struggling for definition by 1980 and just beginning to take shape by 1985. Again, note that this is *not* a terminal technology problem. Consequently, the EFT Special Purpose Terminal product will still be quite young in 1988[23] and by then the EFT function will be fully integrated into the retail POS terminal and is not expected to exist as a separate entity.

Many of the forces (high rate of change, rising personnel costs, energy, etc.) that contribute to the demand found in retail and financial institutions are present in other transaction environments as well. Therefore, a similar continuing demand is expected in factory data collection, hospital patient administration, hotel and airline reservation, and other similar situations.

Perhaps the most exciting trends regarding demand are found as we explore the migration of special purpose terminals from the world of transaction control, to the environment of office automation,[24] to the surroundings of our own homes. Here, the rising costs of energy are continuing to raise the value associated with moving electrons instead of moving physical media or people.[25] Interestingly enough, consumer infatuation with calculators, home computers, and TV games could thrust the Special Purpose Terminal into the home more quickly than will be the case for the office. In fact, this phenomenon will tend to blur the distinction between what is office and what is home. One can foresee a complete, home-based, Special Purpose Terminal system with one or more stations for entertainment, for education, for work, and for shopping and banking transactions. Furthermore, such a system could control the home environment and optimize energy consumption in its spare time.

It is reasonable to expect that such equipment will enter the home first on the entertainment motive and then incorporate the other functions incrementally as the necessary supporting systems (i.e., EFT, education, etc.) come into being. It is projected that by 1985, the bulk of the terminal equipment will be available but the community-level supporting systems will be very primitive. By 1988 the distributed, home-based terminal environment will have positioned itself for growth on into the 1990's.

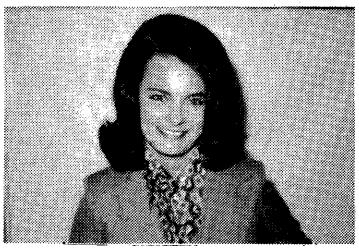
CONCLUSION

We have painted a picture of rising demand and of continuing technological advance which would seem to imply dramatic growth. However, this is offset in those instances where substantial utility-like supporting systems are needed which must cross institutional lines. It doesn't do any good to possess a very sophisticated person-machine interface terminal if there is no distribution system. Therefore, growth is expected; it will be exciting, but except for those pockets of explosive growth that can move ahead on an isolated system basis, the progress *will be slower* than we might otherwise expect. Nevertheless, the Special Purpose Terminal world of 1988 will offer a substantial marketplace for the producer, powerful information systems for the business user, and some real, quality-of-life benefits for the consumer. One of the great entertainment values of the 1980's is going to consist of following the progress of the Special Purpose Terminal. Be sure not to miss it.

REFERENCES

1. "IBM Laser Scanner in Supermarket POS Bow," *Electronic News,* October 13, 1973.
2. Lapides, Gerald, "MOS/LSI Launches the Low Cost Processor," *IEEE Spectrum,* November 1972, pp. 33-40.
3. Featheringham, Tom, "Teleconferences: The Message is the Meeting," *Data Communications,* July 1977, pp. 37-41.
4. Petritz, Richard L., "The Pervasive Microprocessor: Trends and Prospects," *IEEE Spectrum,* July 1977, pp. 18-24.
5. Noyce, Robert N., "Microelectronics," *Scientific American,* September 1977, pp. 63-69.
6. Pullen, Edward W. and Robert G. Simko, "Our Changing Industry," *Datamation,* January 1977, pp. 49-55.
7. Monrad-Krohn, L., "The Micro vs the Minicomputer," *Mini-Micro Systems,* February 1977, pp. 28-33.
8. Bowers, D. M., "Systems-on-a-chip," *Mini-Micro Systems,* May 1976, pp. 74-88.
9. "Semiconductors Face the '80's," *IEEE Spectrum,* October 1977, pp. 42-48.
10. Altman, Lawrence, "Five Technologies Squeezing More Performance From LSI Chips," *Electronics,* August 18, 1977, pp. 91-93.
11. "Computers Poised for Progress," *IEEE Spectrum,* January 1976, pp. 44-49.
12. "Magnetic Bubble Memories," *Digital Design,* May 1977, pp. 50-66.
13. Ross, E. A., "Bubbles vs Floppies," *Mini-Micro Systems,* April 1977, pp. 42-46.
14. Aichroth, Joseph W., "Flat Panel Displays Offer Graphics Alternatives," *Computer Design,* October 1977, pp. 101-106.
15. Bowers, Dan M., "Printers and Teleprinters," *Mini-Micro Systems,* January 1977, pp. 30-53.
16. Buehner, W. L., et al, "Application of Ink Jet Technology to a Word Processing Output Printer," *IBM Journal of Research and Development,* Vol. 21, No. 1, 1977, pp. 2-9.
17. "Worldwide Forecast Shows 12 percent Growth for Year Ahead," *Electronics,* January 6, 1977, pp. 81-104.
18. "J. C. Penney, With 23,000 (POS) Terminals Watching EFT Closely," *Electronic News,* June 23, 1977.
19. McCracken, Daniel D., "Will an Electronic Fund Transfer Network be in the Publics' Interest?," *Data Communication,* April 1977, pp. 59-64.
20. "EFT in the United States," *The Final Report of the National Commission on Electronic Fund Transfers,* October 28, 1977.
21. "Electronic Banking: A Retreat from the Cashless Society," *Business Week,* April 18, 1977, pp. 84-90.
22. "Nebraska Banks Speed EFT Plan Despite Washington Opposition," *Data Communications,* May 1977, p. 14.
23. DeSofi, Oliver J., "EFT in 1988," *Datamation,* January 1977, pp. 63-64.
24. International Data Corporation, "Information Processing and the Office of Tomorrow," *Fortune,* October 1977.
25. "Now FM Broadcasts Carry the Mail," *Business Week,* November 28, 1977.
26. "Hobby Computers," *Datamation,* August 1977, p. 150.
27. Costigan, D. M., "'FAX' in the Home," *IEEE Spectrum,* September 1974, pp. 76-82.

Area Co-Director:
J. F. Rulifson
Xerox Corporation
Palo Alto, California



Area Co-Director:
Donna Williams
Office Systems Integration
Newport Beach, California

Office automation

Daily, new uses appear for computers in the general office environment. While the area is still dominated by word processing, other applications are becoming widespread. Special systems for administrative tasks, records management, document distribution, and communication integrate new activities into the standard word processing and data processing activities.

With these new applications, however, come endless problems—discovering and refining the applications, choosing equipment from the constant new offerings, devising appropriate accounting methods, restructuring organizations, and determining the impact on the corporation—all are complex issues.

The first session will review the history and current state of office automation. The evolutionary changes of the field from the technical and marketing point of view will be reviewed. The history of the industry will be developed and will be compared to the data processing industry. Finally, the surprising diversity of the office automation industry will be outlined with a review of the range and applications and the present areas of concentration.

The architecture of systems for office automation is diverse and rapidly changing. The second session will present three views of these architectural changes; firmware to software, stand-alone to distributed, and maxi to micro. While we observed these changes initially in hardware, they greatly impact the applications. The long range user effects will be the focus of the session.

The third session will address organizational trends in office automation, and will cover methodologies for designing automated office support. They include: integration into computer-based, multi-function terminals for total automated support, cost/benefit analysis for managerial work, increased user participation in the design and revision of automated office systems, and increased attention to organizational and human factors in system design and implementation.

Even with these methodologies, however, the implementation of automated office systems has been difficult. Cost-benefit data is confusing, and overall productivity gains are difficult to determine. This fourth session will present a different, controversial position.

Office automation systems imposed on existing organizational information flows will increase inter-group conflict, confuse assignments, cloud responsibility and provide no overall improvement in performance. If office automation only provides faster switching of messages and easier access to files it will increase

the problems inherent in current information flows and be rejected by the users. What is needed is formal reconstituting of information flows by the major activities of data collection, data base analysis, information packaging and knowledge acceptance. The delineation of basic processes will make possible scheduling and planning models, resource allocation, and assignment of overall objectives and responsibility. Unless these improvements are made in information processes office automation will not realize its full potential.

Finally, some users have begun to reconstitute their information flows and address the issues raised in the previous session. This final, fifth, session focuses on several applications of computer-based message and document distribution systems. Emphasis is on implementation approaches and system value within the organizational context, and on the introduction of such systems to prospective users. Case presentations will be made in which three organizations are represented—one multi-national non-profit, one corporate, and one federal government agency. Each has actually implemented a system and will discuss its usage and related impacts.

Area Director:
Robert Balzer
Information Science Institute
Marina del Rey, California

Artificial intelligence

The bulk of current Artificial Intelligence research is focused on methods for achieving goals within partially understood environments. In such environments the bodies of knowledge which exist are quite varied; some are systematic and well validated, others are judgemental. Characteristically, there are no known optimal ways of applying the knowledge in the environment to achieve the desired goals. The challenge of Al is to develop methods for handling problems in such environments and to aid the development of a better understanding of these environments, which may in turn lead to more powerful ways of achieving goals within it.

The development of Artificial Intelligence has been marked by two rather distinct periods. In the first, the basic structure of the field emerged, centered on the notion of general purpose mechanisms and general-purpose problem solvers, theorem provers, and semantic nets. Once the limitations of such general purpose mechanisms were generally recognized, the second period of development began. It was, and is, marked by the attempt to incorporate and effectively utilize special purpose knowledge appropriate for some task. Issues such as acquiring such special purpose knowledge from experts, representing it in a useful form, and effectively using it to control a problem solving activity became paramount.

These NCC sessions will focus on this latter period through two invited papers. The first, presented by Professor Edward Feigenbaum, chronicles the development of the specialized knowledge period through the activities of the Stanford Heuristic Programming Project which has long been a leader in the development of expert-level performance programs. The second invited paper is the lecture presented by Professor Douglas Lenat upon his selection for the 1977 Computers and Thought Lecture for outstanding contributions by a young researcher to Artificial Intelligence. Lenat's talk and research describe a system for discovering new mathematical concepts and conjectures involving those concepts. This fascinating work indicates the potential feasibility of providing operational models of the process of mathematical discovery.

These two invited papers, which comprise the first Al session, will surely initiate a rash of questions. Therefore, they will immediately be followed by a

session entirely devoted to answering questions from the audience on these and other Al issues. Joining Feigenbaum and Lenat on the panel of Al experts will be Professors Raj Reddy and Saul Amarel, and Dr. Peter Hart. The final session will present several state-of-the-art applications which have achieved expert-level performance.

The art of artificial intelligence—Themes and case studies of knowledge engineering

by EDWARD A. FEIGENBAUM
Stanford University
Stanford, California

INTRODUCTION—AN EXAMPLE

This paper will examine emerging themes of knowledge engineering, illustrate them with case studies drawn from the work of the Stanford Heuristic Programming Project, and discuss general issues of knowledge engineering art and practice.

Let me begin with an example new to our workbench: a system called PUFF, the early fruit of a collaboration between our project and a group at the Pacific Medical Center (PMC) in San Francisco.*

A physician refers a patient to PMC's pulmonary function testing lab for diagnosis of possible pulmonary function disorder. For one of the tests, the patient inhales and exhales a few times in a tube connected to an instrument/computer combination. The instrument acquires data on flow rates and volumes, the so-called flow-volume loop of the patient's lungs and airways. The computer measures certain parameters of the curve and presents them to the diagnostician (physician or PUFF) for interpretation. The diagnosis is made along these lines: normal or diseased; restricted lung disease or obstructive airways disease or a combination of both; the severity; the likely disease type(s) (e.g., emphysema, bronchitis, etc.); and other factors important for diagnosis.

PUFF is given not only the measured data but also certain items of information from the patient record, e.g., sex, age, number of pack-years of cigarette smoking. The task of the PUFF system is to infer a diagnosis and print it out in English in the normal medical summary form of the interpretation expected by the referring physician.

Everything PUFF knows about pulmonary function diagnosis is contained in (currently) 55 rules of the IF. . . THEN. . . form. No textbook of medicine currently records these rules. They constitute the partly-public, partly-private knowledge of an expert pulmonary physiologist at PMC, and were extracted and polished by project engineers working intensively with the expert over a period of time. Here is an example of a PUFF rule (the unexplained acronyms refer to various data measurements):

RULE 31

IF:
1) The severity of obstructive airways disease of the patient is greater than or equal to mild, and
2) The degree of diffusion defect of the patient is greater than or equal to mild, and
3) The tlc (body box) observed/predicted of the patient is greater than or equal to 110 and
4) The observed-predicted difference in rv/tlc of the patient is greater than or equal to 10

THEN:
1) There is strongly suggestive evidence (.9) that the subtype of obstructive airways disease is emphysema, and
2) It is definite (1.0) that "OAD, Diffusion Defect, elevated TLC, and elevated RV together indicate emphysema." is one of the findings.

One hundred cases, carefully chosen to span the variety of disease states with sufficient exemplary information for each, were used to extract the 55 rules. As the knowledge emerged, it was represented in rule form, added to the system and tested by running additional cases. The expert was sometimes surprised, sometimes frustrated, by the occasional gaps and inconsistencies in the knowledge, and the incorrect diagnoses that were logical consequences of the existing rule set. The interplay between knowledge engineer and expert gradually expanded the set of rules to remove most of these problems.

As cumulation of techniques in the art demands and allows, a new tool was not invented when an old one would do. The knowledge engineers pulled out of their toolkit a version of the MYCIN system (to be discussed later), with the rules about infectious diseases removed, and used it as the inference engine for the PUFF diagnoses. Thus PUFF, like MYCIN, is a relatively simple backward-chaining infer-

* Dr. J. Osborn, Dr. R. Fallat, John Kunz, Diane McClung.

ence system. It seeks a valid line-of-reasoning based on its rules and rooted in the instrument and patient data. With a little more work at fitting some existing tools together, PUFF will be able to explain this line-of-reasoning, just as MYCIN does.

As it is, PUFF only prints out the final interpretation, of which the following is an example:

PATIENT DATA:
The degree of dyspnea: MODERATELY-SEVERE
The severity of coughing: MILD
Sputum production MODERATELY-SEVERE
The number of pack-years of smoking: 48
referral diagnosis: BRONCHITIS
IVC/IVC-predicted: 80
RV/RV-predicted: 191
FVC/FVC-predicted: 87
TLC(body box)observed/predicted: 127
Predicted FEV1/FVC: 83
TLC(DLCO)observed/predicted: 83
FEV1/FVC ratio: 50
RV/TLC Observed-Predicted: 21
MMF/MMF-predicted: 19
the DLCO/DLCO-predicted: 48
The slope (F50_obs-F25_obs)/FVC_obs: 19

DEGREE OF OBSTRUCTIVE AIRWAYS DISEASE:

OAD degree by SLOPE: (MODERATELY-SEVERE 700)
OAD degree by MMF: (SEVERE 900)
OAD degree by FEV1: (MODERATELY-SEVERE 700)
FINAL OAD DEGREE: (MODERATELY-SEVERE 910) (SEVERE 900)
No conflict. Final degree: (MODERATELY-SEVERE 910)

INTERPRETATION:

Obstruction is indicated by curvature of the flow-volume loop.
Forced Vital Capacity is normal and peak flow rates are reduced, suggesting airway obstruction.
Flow rate from 25-75 of expired volume is reduced, indicating severe airway obstruction.
OAD, Diffusion Defect, elevated TLC, and elevated RV together indicate emphysema.
OAD, Diffusion Defect, and elevated RV indicate emphysema.
Change in expired flow rates following bronchodilation shows that there is reversibility of airway obstruction.
The presence of a productive cough is an indication that the OAD is of the bronchitic type.
Elevated lung volumes indicate overinflation.
Air trapping is indicated by the elevated difference between observed and predicted RV/TLC ratios.
Improvement in airway resistance indicates some reversibility of airway
Airway obstruction is consistent with the patient's smoking history.
The airway obstruction accounts for the patient's dyspnea.
Although bronchodilators were not useful in this one case, prolonged use may prove to be beneficial to the patient.
The reduced diffusion capacity indicates airway obstruction of the mixed bronchitic and emphysematous types.
Low diffusing capacity indicates loss of alveolar capillary surface.
Obstructive Airways Disease of mixed types

150 cases not studied during the knowledge acquisition process were used for a test and validation of the rule set. PUFF inferred a diagnosis for each. PUFF-produced and expert-produced interpretations were coded for statistical analysis to discover the degree of agreement. Over various types of disease states, and for two conditions of match between human and computer diagnoses ("same degree of severity" and "within one degree of severity"), agreement ranged between approximately 90 percent and 100 percent.

The PUFF story is just beginning and will be told perhaps at a later NCC. The surprising punchline to my synopsis is that the current state of the PUFF system as described above was achieved in less than 50 hours of interaction with the expert and less than 10 man-weeks of effort by the knowledge engineers. We have learned much in the past decade of the art of engineering knowledge-based intelligent agents!

In the remainder of this essay, I would like to discuss the route that one research group, the Stanford Heuristic Programming Project, has taken, illustrating progress with case studies, and discussing themes of the work.

ARTIFICIAL INTELLIGENCE & KNOWLEDGE ENGINEERING

The dichotomy that was used to classify the collected papers in the volume *Computers and Thought* still characterizes well the motivations and research efforts of the AI community. First, there are some who work toward the construction of intelligent artifacts, or seek to uncover principles, methods, and techniques useful in such construction. Second, there are those who view artificial intelligence as (to use Newell's phrase) "theoretical psychology," seeking explicit and valid information processing models of human thought.

For purposes of this essay, I wish to focus on the motivations of the first group, these days by far the larger of the two. I label these motivations "the intelligent agent viewpoint" and here is my understanding of that viewpoint:

"The potential uses of computers by people to accom-

plish tasks can be 'one-dimensionalized' into a spectrum representing the nature of instruction that must be given the computer to do its job. Call it the WHAT-to-HOW spectrum. At one extreme of the spectrum, the user supplies his intelligence to instruct the machine with precision exactly HOW to do his job, step-by-step. Progress in Computer Science can be seen as steps away from the extreme 'HOW' point on the spectrum: the familiar panoply of assembly languages, subroutine libraries, compilers, extensible languages, etc. At the other extreme of the spectrum is the user with his real problem (WHAT he wishes the computer, as his instrument, to do for him). He aspires to communicate WHAT he wants done in a language that is comfortable to him (perhaps English); via communication modes that are convenient for him (including perhaps, speech or pictures); with some generality, some vagueness, imprecision, even error; without having to lay out in detail all necessary subgoals for adequate performance—with reasonable assurance that he is addressing an intelligent agent that is using knowledge of his world to understand his intent, to fill in his vagueness, to make specific his abstractions, to correct his errors, to discover appropriate subgoals, and ultimately to translate WHAT he really wants done into processing steps that define HOW it shall be done by a real computer. The research activity aimed at creating computer programs that act as "intelligent agents" near the WHAT end of the WHAT-To-HOW spectrum can be viewed as the long-range goal of AI research." (Feigenbaum, 1974)

Our young science is still more art than science. Art: "the principles or methods governing any craft or branch of learning." Art: "skilled workmanship, execution, or agency." These the dictionary teaches us. Knuth tells us that the endeavor of computer programming is an art, in just these ways. The art of constructing intelligent agents is both part of and an extension of the programming art. It is the art of building complex computer programs that represent and reason with knowledge of the world. Our art therefore lives in symbiosis with the other worldly arts, whose practitioners—experts of their art—hold the knowledge we need to construct intelligent agents. In most "crafts or branches of learning" what we call "expertise" is the essence of the art. And for the domains of knowledge that we touch with our art, it is the "rules of expertise" or the rules of "good judgment" of the expert practitioners of that domain that we seek to transfer to our programs.

Lessons of the past

Two insights from previous work are pertinent to this essay.

The first concerns the quest for generality and power of the inference engine used in the performance of intelligent acts (what Minsky and Papert [see Goldstein and Papert, 1977] have labeled "the power strategy"). We must hypothesize from our experience to date that the problem solving power exhibited in an intelligent agent's performance is primarily a consequence of the specialist's knowledge employed by the agent, and only very secondarily related to the generality and power of the inference method employed. Our agents must be knowledge-rich, even if they are methods-poor. In 1970, reporting the first major summary-of-results of the DENDRAL program (to be discussed later), we addressed this issue as follows:

> ". . . general problem-solvers are too weak to be used as the basis for building high-performance systems. The behavior of the best general problem-solvers we know, human problem-solvers, is observed to be weak and shallow, except in the areas in which the human problem-solver is a specialist. And it is observed that the transfer of expertise between specialty areas is slight. A chess master is unlikely to be an expert algebraist or an expert mass spectrum analyst, etc. In this view, the expert is the specialist, with a specialist's knowledge of his area and a specialist's methods and heuristics." (Feigenbaum, Buchanan and Lederberg, 1971, p. 187)

Subsequent evidence from our laboratory and all others has only confirmed this belief.

AI researchers have dramatically shifted their view on generality and power in the past decade. In 1967, the canonical question about the DENDRAL program was: "It sounds like good chemistry, but what does it have to do with AI?" In 1977, Goldstein and Papert write of a paradigm shift in AI:

> "Today there has been a shift in paradigm. The fundamental problem of understanding intelligence is not the identification of a few powerful techniques, but rather the question of how to represent large amounts of knowledge in a fashion that permits their effective use and interaction." (Goldstein and Papert, 1977).

The second insight from past work concerns the nature of the knowledge that an expert brings to the performance of a task. Experience has shown us that this knowledge is largely heuristic knowledge, experiential, uncertain—mostly "good guesses" and "good practice," in lieu of facts and rigor. Experience has also taught us that much of this knowledge is private to the expert, not because he is unwilling to share publicly how he performs, but because he is unable. He knows more than he is aware of knowing. [Why else is the Ph.D. or the Internship a guild-like apprenticeship to a presumed "master of the craft?" What the masters really know is not written in the textbooks of the masters.] But we have learned also that this private knowledge can be uncovered by the careful, painstaking analysis of a second party, or sometimes by the expert himself, operating in the context of a large number of highly specific performance problems. Finally, we have learned that expertise is multifaceted, that the expert brings to bear many and varied sources of knowledge in performance. The approach to capturing his expertise must proceed on many fronts simultaneously.

The knowledge engineer

The knowledge engineer is that second party just discussed. She works intensively with an expert to acquire domain-specific knowledge and organize it for use by a program. Simultaneously she is matching the tools of the AI workbench to the task at hand—program organizations, methods of symbolic inference, techniques for the structuring of symbolic information, and the like. If the tool fits, or nearly fits, she uses it. If not, necessity mothers AI invention, and a new tool gets created. She builds the early versions of the intelligent agent, guided always by her intent that the program eventually achieve expert levels of performance in the task. She refines or reconceptualizes the system as the increasing amount of acquired knowledge causes the AI tool to "break" or slow down intolerably. She also refines the human interface to the intelligent agent with several aims: to make the system appear "comfortable" to the human user in his linguistic transactions with it; to make the system's inference processes understandable to the user; and to make the assistance controllable by the user when, in the context of a real problem, he has an insight that previously was not elicited and therefore not incorporated.

In the next section, I wish to explore (in summary form) some case studies of the knowledge engineer's art.

CASES FROM THE KNOWLEDGE ENGINEER'S WORKSHOP

I will draw material for this section from the work of my group at Stanford. Much exciting work in knowledge engineering is going on elsewhere. Since my intent is not to survey literature but to illustrate themes, at the risk of appearing parochial I have used as case studies the work I know best.

My collaborators (Professors Lederberg and Buchanan) and I began a series of projects, initially the development of the DENDRAL program, in 1965. We had dual motives: first, to study scientific problem solving and discovery, particularly the processes scientists do use or should use in inferring hypotheses and theories from empirical evidence; and second, to conduct this study in such a way that our experimental programs would one day be of use to working scientists, providing intelligent assistance on important and difficult problems. By 1970, we and our co-workers had gained enough experience that we felt comfortable in laying out a program of research encompassing work on theory formation, knowledge utilization, knowledge acquisition, explanation, and knowledge engineering techniques. Although there were some surprises along the way, the general lines of the research are proceeding as envisioned.

THEMES

As a road map to these case studies, it is useful to keep in mind certain major themes:

Generation-and-test: Omnipresent in our experiments is the "classical" generation-and-test framework that has been the hallmark of AI programs for two decades. This is not a consequence of a doctrinaire attitude on our part about heuristic search, but rather of the usefulness and sufficiency of the concept.

Situation⇒Action Rules: We have chosen to represent the knowledge of experts in this form. Making no doctrinaire claims for the universal applicability of this representation, we nonetheless point to the demonstrated utility of the rule-based representation. From this representation flow rather directly many of the characteristics of our programs: for example, ease of modification of the knowledge, ease of explanation. The essence of our approach is that a rule must capture a "chunk" of domain knowledge that is meaningful, in and of itself, to the domain specialist. Thus our rules bear only a historical relationship to the production rules used by Newell and Simon (1972) which we view as "machine-language programming" of a recognize⇒act machine.

The Domain-Specific Knowledge: It plays a critical role in organizing and constraining search. The theme is that in the knowledge is the power. The interesting action arises from the knowledge base, not the inference engine. We use knowledge in rule form (discussed above), in the form of inferentially-rich models based on theory, and in the form of tableaus of symbolic data and relationships (i.e., frame-like structures). System processes are made to conform to natural and convenient representations of the domain-specific knowledge.

Flexibility to modify the knowledge base: If the so-called "grain size" of the knowledge representation is chosen properly (i.e., small enough to be comprehensible but large enough to be meaningful to the domain specialist), then the rule-based approach allows great flexibility for adding, removing, or changing knowledge in the system.

Line-of-reasoning: A central organizing principle in the design of knowledge-based intelligent agents is the maintenance of a line-of-reasoning that is comprehensible to the domain specialist. This principle is, of course, not a logical necessity, but seems to us to be an engineering principle of major importance.

Multiple Sources of Knowledge: The formation and maintenance (support) of the line-of-reasoning usually require the integration of many disparate sources of knowledge. The representational and inferential problems in achieving a smooth and effective integration are formidable engineering problems.

Explanation: The ability to explain the line-of-reasoning in a language convenient to the user is necessary for application and for system development (e.g., for debugging and for extending the knowledge base). Once again, this is an engineering principle, but very important. What con-

stitutes "an explanation" is not a simple concept, and considerable thought needs to be given, in each case, to the structuring of explanations.

CASE STUDIES

In this section I will try to illustrate these themes with various case studies.

DENDRAL: Inferring chemical structures

Historical note

Begun in 1965, this collaborative project with the Stanford Mass Spectrometry Laboratory has become one of the longest-lived continuous efforts in the history of AI (a fact that in no small way has contributed to its success). The basic framework of generation-and-test and rule-based representation has proved rugged and extendable. For us the DENDRAL system has been a fountain of ideas, many of which have found their way, highly metamorphosed, into our other projects. For example, our long-standing commitment to rule-based representations arose out of our (successful) attempt to head off the imminent ossification of DENDRAL caused by the rapid accumulation of new knowledge in the system around 1967.

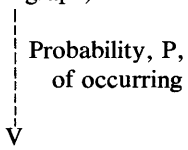
Task

To enumerate plausible structures (atom-bond graphs) for organic molecules, given two kinds of information: analytic instrument data from a mass spectrometer and a nuclear magnetic resonance spectrometer; and user-supplied constraints on the answers, derived from any other source of knowledge (instrumental or contextual) available to the user.

Representations

Chemical structures are represented as node-link graphs of atoms (nodes) and bonds (links). Constraints on search are represented as subgraphs (atomic configurations) to be denied or preferred. The empirical theory of mass spectrometry is represented by a set of rules of the general form:

Situation: Particular atomic configuration (subgraph)



Action: Fragmentation of the particular configuration (Breaking links)

Rules of this form are natural and expressive to mass spectrometrists.

Sketch of method

DENDRAL's inference procedure is a heuristic search that takes place in three stages, without feedback: plan-generate-test.

"Generate" (a program called CONGEN) is a generation process for plausible structures. Its foundation is a combinatorial algorithm (with mathematically proven properties of completeness and non-redundant generation) that can produce all the topologically legal candidate structures. Constraints supplied by the user or by the "Plan" process prune and steer the generation to produce the plausible set (i.e., those satisfying the constraints) and not the enormous legal set.

"Test" refines the evaluation of plausibility, discarding less worthy candidates and rank-ordering the remainder for examination by the user. "Test" first produces a "predicted" set of instrument data for each plausible candidate, using the rules described. It then evaluates the worth of each candidate by comparing its predicted data with the actual input data. The evaluation is based on heuristic criteria of goodness-of-fit. Thus, "test" selects the "best" explanations of the data.

"Plan" produces direct (i.e., not chained) inference about likely substructure in the molecule from patterns in the data that are indicative of the presence of the substructure. (Patterns in the data trigger the left-hand-sides of substructure rules). Though composed of many atoms whose interconnections are given, the substructure can be manipulated as atom-like by "generate." Aggregating many units entering into a combinatorial process into fewer higher-level units reduces the size of the combinatorial search space. "Plan" sets up the search space so as to be relevant to the input data. "Generate is the inference tactician; "Plan" is the inference strategist. There is a separate "Plan" package for each type of instrument data, but each package passes substructures (subgraphs) to "Generate." Thus, there is a uniform interface between "Plan" and "Generate." User-supplied constraints enter this interface, directly or from user-assist packages, in the form of substructures.

Sources of knowledge

The various sources of knowledge used by the DENDRAL system are:

Valences (legal connections of atoms); stable and unstable configurations of atoms; rules for mass spectrometry fragmentations; rules for NMR shifts; experts' rules for planning and evaluation; user-supplied constraints (contextual).

Results

DENDRAL's structure elucidation abilities are, paradoxically, both very general and very narrow. In general, DENDRAL handles all molecules, cyclic and tree-like. In pure structure elucidation under constraints (without instrument data), CONGEN is unrivaled by human performance. In structure elucidation with instrument data, DENDRAL's performance rivals expert human performance only for a small number of molecular families for which the program has been given specialist's knowledge, namely the families of interest to our chemist collaborators. I will spare this computer science audience the list of names of these families. Within these areas of knowledge-intensive specialization, DENDRAL's performance is usually not only much faster but also more accurate than expert human performance.

The statement just made summarizes thousands of runs of DENDRAL on problems of interest to our experts, their colleagues, and their students. The results obtained, along with the knowledge that had to be given to DENDRAL to obtain them, are published in major journals of chemistry. To date, 25 papers have been published there, under a series title "Applications of Artificial Intelligence for Chemical Inference: ⟨specific subject⟩" (see for example, the Buchanan, Smith, et al., 1976, reference).

The DENDRAL system is in everyday use by Stanford chemists, their collaborators at other universities and collaborating or otherwise interested chemists in industry. Users outside Stanford access the system over commercial computer/communications network. The problems they are solving are often difficult and novel. The British government is currently supporting work at Edinburgh aimed at transferring DENDRAL to industrial user communities in the UK.

Discussion

Representation and extensibility. The representation chosen for the molecules, constraints, and rules of instrument data interpretation is sufficiently close to that used by chemists in thinking about structure elucidation that the knowledge base has been extended smoothly and easily, mostly by chemists themselves in recent years. Only one major reprogramming effort took place in the last 9 years—when a new generator was created to deal with cyclic structures.

Representation and the Integration of multiple sources of knowledge. The generally difficult problem of integrating various sources of knowledge has been made easy in DENDRAL by careful engineering of the representations of objects, constraints, and rules. We insisted on a common language of compatibility of the representations with each other and with the inference processes: the language of molecular structure expressed as graphs. This leads to a straightforward procedure for adding a new source of knowledge, say, for example, the knowledge associated with a new type of instrument data. The procedure is this: write rules that describe the effect of the physical processes of the instrument on molecules using the situation⇒action form with molecular graphs on both sides; any special inference process using these rules must pass its results to the generator only (!) in the common graph language.

It is today widely believed in AI that the use of many diverse sources of knowledge in problem solving and data interpretation has a strong effect on quality of performance. How strong is, of course, domain-dependent, but the impact of bringing just one additional source of knowledge to bear on a problem can be startling. In one difficult (but not unusually difficult) mass spectrum analysis problem,* the program using its mass spectrometry knowledge alone would have generated an impossibly large set of plausible candidates (over 1.25 million!). Our engineering response to this was to add another source of data and knowledge, proton NMR. The addition on a simple interpretive theory of this NMR data, from which the program could infer a few additional constraints, reduced the set of plausible candidates to one, the right structure! This was not an isolated result but showed up dozens of times in subsequent analyses.

DENDRAL and data. DENDRAL's robust models (topological, chemical, instrumental) permit a strategy of finding solutions by generating hypothetical "correct answers" and choosing among these with critical tests. This strategy is opposite to that of piecing together the implications of each data point to form a hypothesis. We call DENDRAL's strategy largely model-driven, and the other data-driven. The consequence of having enough knowledge to do model-driven analysis is a large reduction in the amount of data that must be examined since data is being used mostly for verification of possible answers. In a typical DENDRAL mass spectrum analysis, usually no more than about 25 data points out of a typical total of 250 points are processed. This important point about data reduction and focus-of-attention has been discussed before by Gregory (1968) and by the vision and speech research groups, but is not widely understood.

Conclusion. DENDRAL was an early herald of AI's shift to the knowledge-based paradigm. It demonstrated the point of the primacy of domain-specific knowledge in achieving expert levels of performance. Its development brought to the surface important problems of knowledge representation, acquisition, and use. It showed that, by and large, the AI tools of the first decade were sufficient to cope with the demands of a complex scientific problem-solving task, or were readily extended to handle unforeseen difficulties. It demonstrated that AI's conceptual and programming tools were capable of producing programs of applications interest, albeit in narrow specialties. Such a demonstration of competence and sufficiency was important for the credibility of the AI field at a critical juncture in its history.

META-DENDRAL: inferring rules of mass spectrometry

Historical note

The META-DENDRAL program is a case study in automatic acquisition of domain knowledge. It arose out of our

* The analysis of an acyclic amine with formula C20H45N.

DENDRAL work for two reasons: first, a decision that with DENDRAL we had a sufficiently firm foundation on which to pursue our long-standing interest in processes of scientific theory formation; second, by a recognition that the acquisition of domain knowledge was the bottleneck problem in the building of applications-oriented intelligent agents.

Task

META-DENDRAL's job is to infer rules of fragmentation of molecules in a mass spectrometer for possible later use by the DENDRAL performance program. The inference is to be made from actual spectra recorded from known molecular structures. The output of the system is the set of fragmentation rules discovered, summary of the evidence supporting each rule, and a summary of contra-indicating evidence. User-supplied constraints can also be input to force the form of rules along desired lines.

Representations

The rules are, of course, of the same form as used by DENDRAL that was described earlier.

Sketch of method

META-DENDRAL, like DENDRAL, uses the generation-and-test framework. The process is organized in three stages: Reinterpret the data and summarize evidence (INTSUM); generate plausible candidates for rules (RULEGEN); test and refine the set of plausible rules (RULEMOD).

INTSUM: gives every data point in every spectrum an interpretation as a possible (highly specific) fragmentation. It then summarizes statistically the "weight of evidence" for fragmentations and for atomic configurations that cause these fragmentations. Thus, the job of INTSUM is to translate data to DENDRAL subgraphs and bond-breaks, and to summarize the evidence accordingly.

RULEGEN: conducts a heuristic search of the space of all rules that are legal under the DENDRAL rule syntax and the user-supplied constraints. It searches for plausible rules, i.e., those for which positive evidence exists. A search path is pruned when there is no evidence for rules of the class just generated. The search tree begins with the (single) most general rule (loosely put, "anything" fragments from "anything") and proceeds level-by-level toward more detailed specifications of the "anything." The heuristic stopping criterion measures whether a rule being generated has become too specific, in particular whether it is applicable to too few molecules of the input set. Similarly there is a criterion for deciding whether an emerging rule is too general. Thus, the output of RULEGEN is a set of candidate rules for which there is positive evidence.

RULEMOD: tests the candidate rule set using more complex criteria, including the presence of negative evidence. It removes redundancies in the candidate rule set; merges rules that are supported by the same evidence; tries further specialization of candidates to remove negative evidence; and tries further generalization that preserves positive evidence.

Results

META-DENDRAL produces rule sets that rival in quality those produced by our collaborating experts. In some tests, META-DENDRAL re-created rule sets that we had previously acquired from our experts during the DENDRAL project. In a more stringent test involving members of a family of complex ringed molecules for which the mass spectral theory had not been completely worked out by chemists, META-DENDRAL discovered rule sets for each subfamily. The rules were judged by experts to be excellent and a paper describing them was recently published in a major chemical journal (Buchanan, Smith, et al, 1976).

In a test of the generality of the approach, a version of the META-DENDRAL program is currently being applied to the discovery of rules for the analysis of nuclear magnetic resonance data.

MYCIN and TEIRESIAS: Medical diagnosis

Historical note

MYCIN originated in the Ph.D. thesis of E. Shortliffe (now Shortliffe, M.D. as well), in collaboration with the Infectious Disease group at the Stanford Medical School (Shortliffe, 1976). TEIRESIAS, the Ph.D. thesis work of R. Davis, arose from issues and problems indicated by the MYCIN project but generalized by Davis beyond the bounds of medical diagnosis applications (Davis, 1976). Other MYCIN-related theses are in progress.

Tasks

The MYCIN performance task is diagnosis of blood infections and meningitis infections and the recommendation of drug treatment. MYCIN conducts a consultation (in English) with a physician-user about a patient case, constructing lines-of-reasoning leading to the diagnosis and treatment plan.

The TEIRESIAS knowledge acquisition task can be described as follows:

In the context of a particular consultation, confront the expert with a diagnosis with which he does not agree. Lead him systematically back through the line-of-reasoning that produced the diagnosis to the point at which he indicates the analysis went awry. Interact with the expert to modify offending rules or to acquire new rules. Rerun the consultation to test the solution and gain the expert's concurrence.

Representations:

MYCIN's rules are of the form:

IF ⟨conjunctive clauses⟩ THEN ⟨implication⟩

Here is an example of a MYCIN rule for blood infections.

RULE 85

IF:
1) The site of the culture is blood, and
2) The gram stain of the organism is gramneg, and
3) The morphology of the organism is rod, and
4) The patient is a compromised host

THEN:
There is suggestive evidence (.6) that the identity of the organism is pseudomonas-aeruginosa

TEIRESIAS allows the representation of MYCIN-like rules governing the use of other rules, i.e., rule-based strategies. An example follows.

METARULE 2

IF:
1) the patient is a compromised host, and
2) there are rules which mention in their premise pseudomonas
3) there are rules which mention in their premise klebsiellas

THEN:
There is suggestive evidence (.4) that the former should be done before the latter.

Sketch of method

MYCIN employs a generation-and-test procedure of a familiar sort. The generation of steps in the line-of-reasoning is accomplished by backward chaining of the rules. An IF-side clause is either immediately true or false (as determined by patient or test data entered by the physician in the consultation); or is to be decided by subgoaling. Thus, "test" is interleaved with "generation" and serves to prune out incorrect lines-of-reasoning.

Each rule supplied by an expert has associated with it a "degree of certainty" representing the expert's confidence in the validity of the rule (a number from 1 to 10). MYCIN uses a particular ad-hoc but simple model of inexact reasoning to cumulate the degrees of certainty of the rules used in an inference chain (Shortliffe and Buchanan, 1975).

It follows that there may be a number of "somewhat true" lines-of-reasoning—some indicating one diagnosis, some indicating another. All (above a threshold) are used by the system as sources of knowledge indicating plausible lines-of-reasoning.

TEIRESIAS' rule acquisition process is based on a record of MYCIN's search. Rule acquisition is guided by a set of rule models that dictate the form and indicate the likely content of new rules. Rule models are not given in advance, but are inferred from the knowledge base of existing rules.

Language of interaction

The language used looks like it might be English but is actually the dialect "Doctor-ese" used by members of the tribe of healing arts practitioners. Reasonably simple language processing methods suffice. When ambiguities of interpretation are encountered, they are fed back to the user for decisions about meaning.

Example diagnosis and treatment plan

The following is an example of a diagnosis reported by MYCIN to one of its users following a consultation. An example consultation will not be shown, but one can be found in Shortliffe's book (Shortliffe, 1976).

My therapy recommendations will be designed to treat for organisms that are either very likely or, although less likely, would have a significant effect on therapy selection if they were present. It is important to cover for the following probable infection(s) and associated organism(s):

INFECTION-1 is CYSTITIS
⟨Item 1⟩ PSEUDOMONAS-CEPACIA [ORGANISM-5]
⟨Item 2⟩ CITROBACTER-DIVERSUS [ORGANISM-4]
⟨Item 3⟩ E.COLI [ORGANISM-3]

INFECTION-2 is UPPER-RESPIRATORY-INFECTION
⟨Item 4⟩ LISTERIA [ORGANISM-2]

INFECTION-3 is BACTEREMIA
⟨Item 5⟩ ENTEROCOCCUS [ORGANISM-1]

[REC-1] My preferred therapy recommendation is as follows:

In order to cover for Items ⟨1 2 3 4 5⟩:
Give the following in combination:
1) KANAMYCIN
Dose: 750 mg (7.5 mg/kg) q12h IM (or IV) for 28 days
Comments: Modify dose in renal failure
2) PENICILLIN
Dose: 2,500,000 units (25000 units/kg) q4h IV for 28 days

Results

No clinical evaluation of MYCIN is currently being done. In a preliminary evaluation, done outside of the clinical setting, a panel of expert judges was asked to rate MYCIN's performance along a number of dimensions (e.g., infecting organism identification, organism significance). In 90 percent of the cases submitted to the judges, a majority of the judges said that the program's decisions were the-same-as or as-good-as the decisions they would have made.

Explanation

MYCIN can expose its line of reasoning by offering explanations of various kinds. "Why" and "How" questions can be asked by the user during the consultation. "Why" is the user's way of inquiring, "Why do you want to know that?" "How" is his way of asking, "How did you conclude that. . . ?" In a post-consultation, he can similarly inquire about how the diagnostic conclusions were reached, and inquire about the reasons why certain other conclusions were ruled out. Examples of the MYCIN-TEIRESIAS explanation capabilities follow:

MYCIN: Is the patient's illness with ORGANISM-1 a hospital-acquired infection?
USER: WHY
TEIRESIAS: (i.e., WHY is it important to determine whether or not the infection with ORGANISM-1 was acquired while the patient was hospitalized?)

[1.0] This will aid in determining the category of ORGANISM-1.

It has already been established that

[1.1] the gram stain of ORGANISM-1 is gramneg, and
[1.2] the morphology of ORGANISM-1 is rod, and
[1.3] the aerobicity of ORGANISM-1 is facultative

Therefore, if

[1.4] the infection with ORGANISM-1 was not acquired while the patient was hospitalized

then

there is strongly suggestive evidence (.8) that the category of ORGANISM-1 is enterobacteriaceae

Also: there is weakly suggestive evidence (.1) that the identity of ORGANISM-1 is pseudomonas-aeruginosa [RULE 37]

USER: HOW 1.3

TEIRESIAS: [i.e. HOW was it established that the aerobicity of ORGANISM-1 is facultative?]

The following were used:

[3.1] RULE 27 indicated there is strongly suggestive evidence (.8) that the aerobicity of ORGANISM-1 is facultative

[3.2] RULE 27 indicated there is weakly suggestive evidence (.2) that the aerobicity of ORGANISM-1 is anaerobic

Since this gave a cumulative CF of (.8) for facultative, and (.2) for anaerobic, it has been established that the aerobicity of ORGANISM-1 is facultative.

USER: WHY DIDN'T YOU CONSIDER STREPTOCOCCUS AS A POSSIBILITY FOR ORGANISM-1?

TEIRESIAS: The following rule could have been used to determine that the identity of ORGANISM-1 was streptococcus: RULE 33

But clause 2 ("the morphology of the organism is coccus") was already known to be false for ORGANISM-1, so the rule was never tried.

Discussion

A few short comments aimed at knowledge engineering are in order.

English is the language most natural and comfortable for our experts and users, and was therefore the language chosen for interactive consultation, explanation, and external representation of the rules (the internal format is INTERLISP). This situation is not peculiar to doctors; in most areas of application of intelligent agents I believe that English (i.e., natural language) will be the language of choice. Programming an English language processor and front-end to such systems is not a scary enterprise because:

(a) the domain is specialized, so that possible interpretations are constrained.

(b) specialist-talk is replete with standard jargon and stereotyped ways of expressing knowledge and queries—just right for text templates, simple grammars and other simple processing schemes.

(c) the ambiguity of interpretation resulting from simple schemes can be dealt with easily by feeding back interpretations for confirmation. If this is done with a pleasant "I didn't quite understand you. . ." tone, it is not irritating to the user.

English may be exactly the wrong language for representation and interaction in some domains. It would be awkward, to say the least, to represent DENDRAL's chemical structures and knowledge of mass spectrometry in English, or to interact about these with a user.

Simple explanation schemes have been a part of the AI scene for a number of years and are not hard to implement. Really good models of what explanation is as a transaction between user and agent, with programs to implement these models, will be the subject (I predict) of much future research in AI.

Without the explanation capability, I assert, user acceptance of MYCIN would have been nil, and there would have been a greatly diminished effectiveness and contribution of our experts.

MYCIN was the first of our programs that forced us to deal with what we had always understood: that experts' knowledge is uncertain and that our inference engines had to be made to reason with this uncertainty. It is less important that the inexact reasoning scheme be formal, rigorous, and uniform than it is for the scheme to be natural to and easily understandable by the experts and users.

All of these points can be summarized by saying that MYCIN and its TEIRESIAS adjunct are experiments in the design of a see-through system, whose representations and processes are almost transparently clear to the domain specialist. "Almost" here is equivalent to "with a few minutes of introductory description." The various pieces of MYCIN—the backward chaining, the English transactions, the explanations, etc.—are each simple in concept and realization. But there are great virtues to simplicity in system design; and viewed as a total intelligent agent system, MYCIN/TEIRESIAS is one of the best engineered.

SU/X: signal understanding

Historical note

SU/X is a system design that was tested in an application whose details are classified. Because of this, the ensuing discussion will appear considerably less concrete and tangible than the preceding case studies. This system design was done by H. P. Nii and me, and was strongly influenced by the CMU Hearsay II system design (Lesser and Erman, 1977).

Task

SU/X's task is the formation and continual updating, over long periods of time, of hypotheses about the identity, location, and velocity of objects in a physical space. The output desired is a display of the "current best hypotheses" with full explanation of the support for each. There are two types of input data: the primary signal (to be understood); and auxiliary symbolic data (to supply context for the understanding). The primary signals are spectra, represented as descriptions of the spectral lines. The various spectra cover the physical space with some spatial overlap.

Representations

The rules given by the expert about objects, their behavior, and the interpretation of signal data from them are all represented in the situation⇒action form. The "situations" constitute invoking conditions and the "actions" are processes that modify the current hypotheses, post unresolved issues, recompute evaluations, etc. The expert's knowledge of how to do analysis in the task is also represented in rule form. These strategy rules replace the normal executive program.

The situation-hypothesis is represented as a node-link graph, tree-like in that it has distinct "levels," each representing a degree of abstraction (or aggregation) that is natural to the expert in his understanding of the domain. A node represents an hypothesis; a link to that node represents support for that hypothesis (as in HEARSAY II, "support from above" or "support from below"). "Lower" levels are concerned with the specifics of the signal data. "Higher" levels represent symbolic abstractions.

Sketch of method

The situation-hypothesis is formed incrementally. As the situation unfolds over time, the triggering of rules modifies or discards existing hypotheses, adds new ones, or changes support values. The situation-hypothesis is a common workspace ("blackboard," in HEARSAY jargon) for all the rules.

In general, the incremental steps toward a more complete and refined situation-hypothesis can be viewed as a sequence of local generate-and-test activities. Some of the rules are plausible move generators, generating either nodes or links. Other rules are evaluators, testing and modifying node descriptions.

In typical operation, new data is submitted for processing (say, N time-units of new data). This initiates a flurry of rule-triggerings and consequently rule-actions (called "events"). Some events are direct consequences of data; other events arise in a cascade-like fashion from the triggering of rules. Auxiliary symbolic data also cause events, usually affecting the higher levels of the hypothesis. As a consequence, support-from-above for the lower level processes is made available; and expectations of possible lower level events can be formed. Eventually all the relevant rules have their say and the system becomes quiescent, thereby triggering the input of new data to reenergize the inference activity.

The system uses the simplifying strategy of maintaining only one "best" situation-hypothesis at any moment, modifying it incrementally as required by the changing data. This

approach is made feasible by several characteristics of the domain. First, there is the strong continuity over time of objects and their behaviors (specifically, they do not change radically over time, or behave radically differently over short periods). Second, a single problem (identity, location and velocity of a particular set of objects) persists over numerous data gathering periods. (Compare this to speech understanding in which each sentence is spoken just once, and each presents a new and different problem.) Finally, the system's hypothesis is typically "almost right," in part because it gets numerous opportunities to refine the solution (i.e., the numerous data gathering periods), and in part because the availability of many knowledge sources tends to over-determine the solution. As a result of all of these, the current best hypothesis changes only slowly with time, and hence keeping only the current best is a feasible approach.

Of interest are the time-based events. These rule-like expressions, created by certain rules, trigger upon the passage of specified amounts of time. They implement various "wait-and-see" strategies of analysis that are useful in the domain.

Results

In the test application, using signal data generated by a simulation program because real data was not available, the program achieved expert levels of performance over a span of test problems. Some problems were difficult because there was very little primary signal to support inference. Others were difficult because too much signal induced a plethora of alternatives with much ambiguity.

A modified SU/X design is currently being used as the basis for an application to the interpretation of x-ray crystallographic data, the CRYSALIS program mentioned later.

Discussion

The role of the auxiliary symbolic sources of data is of critical importance. They supply a symbolic model of the existing situation that is used to generate expectations of events to be observed in the data stream. This allows flow of inferences from higher levels of abstraction to lower. Such a process, so familiar to AI researchers, apparently is almost unrecognized among signal processing engineers. In the application task, the expectation-driven analysis is essential in controlling the combinatorial processing explosion at the lower levels, exactly the explosion that forces the traditional signal processing engineers to seek out the largest possible number-cruncher for their work.

The design of appropriate explanations for the user takes an interesting twist in SU/X. The situation-hypothesis unfolds piecemeal over time, but the "appropriate" explanation for the user is one that focuses on individual objects over time. Thus the appropriate explanation must be synthesized from a history of all the events that led up to the current hypothesis. Contrast this with the MYCIN-TEIRESIAS reporting of rule invocations in the construction of a reasoning chain.

Since its knowledge base and its auxiliary symbolic data give it a model-of-the-situation that strongly constrains interpretation of the primary data stream, SU/X is relatively unperturbed by errorful or missing data. These data conditions merely cause fluctuations in the credibility of individual hypotheses and/or the creation of the "wait-and-see" events. SU/X can be (but has not yet been) used to control sensors. Since its rules specify what types and values of evidence are necessary to establish support, and since it is constantly processing a complete hypothesis structure, it can request "critical readings" from the sensors. In general, this allows an efficient use of limited sensor bandwidth and data acquisition processing capability.

Other case studies

Space does not allow more than just a brief sketch of other interesting projects that have been completed or are in progress.

AM: mathematical discovery

AM is a knowledge-based system that conjectures interesting concepts in elementary mathematics. It is a discoverer of interesting theorems to prove, not a theorem proving program. It was conceived and executed by D. Lenat for his Ph.D. thesis, and is reported by him in these proceedings.

AM's knowledge is basically of two types: rules that suggest possibly interesting new concepts from previously conjectured concepts; and rules that evaluate the mathematical "interestingness" of a conjecture. These rules attempt to capture the expertise of the professional mathematician at the task of mathematical discovery. Though Lenat is not a professional mathematician, he was able successfully to serve as his own expert in the building of this program.

AM conducts a heuristic search through the space of concepts creatable from its rules. Its basic framework is generation-and-test. The generation is plausible move generation, as indicated by the rules for formation of new concepts. The test is the evaluation of "interestingness." Of particular note is the method of test-by-example that lends the flavor of scientific hypothesis testing to the enterprise of mathematical discovery.

Initialized with concepts of elementary set theory, it conjectured concepts in elementary number theory, such as "add," "multiply" (by four distinct paths!), "primes," the unique factorization theorem, and a concept similar to primes but previously not much studied called "maximally divisible numbers."

MOLGEN: planning experiments in molecular genetics

MOLGEN, a collaboration with the Stanford Genetics Department, is work in progress. MOLGEN's task is to

provide intelligent advice to a molecular geneticist on the planning of experiments involving the manipulation of DNA. The geneticist has various kinds of laboratory techniques available for changing DNA material (cuts, joins, insertions, deletions, and so on); techniques for determining the biological consequences of the changes; various instruments for measuring effects; various chemical methods for inducing, facilitating, or inhibiting changes; and many other tools.

Some MOLGEN programs under development will offer planning assistance in organizing and sequencing such tools to accomplish an experimental goal. Other MOLGEN programs will check user-provided experiment plans for feasibility; and its knowledge base will be a repository for the rapidly expanding knowledge of this specialty, available by interrogation.

In MOLGEN the problem of integration of many diverse sources of knowledge is central since the essence of the experiment planning process is the successful merging of biological, genetic, chemical, topological, and instrument knowledge. In MOLGEN the problem of representing processes is also brought into focus since the expert's knowledge of experimental strategies—proto-plans—must also be represented and put to use.

One MOLGEN program (Stefik, 1978) solves a type of analysis problem that is often difficult for laboratory scientists to solve. DNA structures can be fragmented by chemicals called restriction enzymes. These enzymes cut DNA at specific recognition sites. The fragmentation may be complete or partial. One or more enzymes may be used. The fragmented segments of the DNA are collected and sorted out by segment length using a technique called gel electrophoresis. The analytical problem is similar to that faced by DENDRAL: given an observed fragmentation pattern, hypothesize the best structural explanation of the data. More precisely the problem is to map the enzyme recognition sites of a DNA structure from complete or partial "digests".

The program uses the model-driven approach that is similar to DENDRAL's and is discussed earlier. The method is generate-and-test. A generator is initiated that is capable of generating all the site-segment maps in an exhaustive, irredundant fashion. Various pruning rules are used to remove whole classes of conceivable candidates in light of the data. Some of the pruning rules are empirical and judgmental. Others are formal and mathematically based.

The program solves simpler problems of this type of analysis better than laboratory scientists. The harder problems, however, yield only to the broader biological knowledge known by the scientists and not yet available to the program's reasoning processes. In a recent test case, a problem whose solution space contained approximately 150,000,000 site-fragment "maps" was solved in 27 seconds of PDP-10 time using the INTERLISP programming system.

Interestingly, the computer scientist's formal understanding of the nature of the problem, his formal representation of the knowledge used for pruning out inappropriate candidates, and the computational power available to him enabled him to suggest a few new experiment designs to his geneticist collaborators that were not previously in their repertoire.

CRYSALIS: inferring protein structure from electron density maps

CRYSALIS, too, is work in progress. Its task is to hypothesize the structure of a protein from a map of electron density that is derived from x-ray crystallographic data. The map is three-dimensional, and the contour information is crude and highly ambiguous. Interpretation is guided and supported by auxiliary information, of which the amino acid sequence of the protein's backbone is the most important. Density map interpretation is a protein chemist's art. As always, capturing this art in heuristic rules and putting it to use with an inference engine is the project's goal.

The inference engine for CRYSALIS is a modification of the SU/X system design described above. The hypothesis formation process must deal with many levels of possibly useful aggregation and abstraction. For example, the map itself can be viewed as consisting of "peaks," or "peaks and valleys," or "skeleton." The protein model has "atoms," "amide planes," "amino acid sidechains," and even massive substructures such as "helices." Protein molecules are so complex that a systematic generation-and-test strategy like DENDRAL's is not feasible. Incremental piecing together of the hypothesis using region-growing methods is necessary.

The CRYSALIS design (alias SU/P) is described in a recent paper by Nii and Feigenbaum (1977).

SUMMARY OF CASE STUDIES

Some of the themes presented earlier need no recapitulation, but I wish to revisit three here: generation-and-test; situation⇒action rules; and explanations.

Generation and test

Aircraft come in a wide variety of sizes, shapes, and functional designs and they are applied in very many ways. But almost all that fly do so because of the unifying physical principle of lift by airflow; the others are described by exception. If there is such a unifying principle for intelligent programs and human intelligence it is generation-and-test. No wonder that this has been so thoroughly studied in AI research!

In the case studies, generation is manifested in a variety of forms and processing schemes. There are legal move generators defined formally by a generating algorithm (DENDRAL's graph generating algorithm); or by a logical rule of inference (MYCIN's backward chaining). When legal move generation is not possible or not efficient, there are plausible move generators (as in SU/X and AM). Sometimes generation is interleaved with testing (as in MYCIN, SU/X, and AM). In one case, all generation precedes testing (DENDRAL). One case (META-DENDRAL) is mixed, with some testing taking place during generation, some after.

Test also shows great variety. There are simple tests (MYCIN: "Is the organism aerobic?"; SU/X: "Has a spectral line appeared at position P?") Some tests are complex heuristic evaluations (AM: "Is the new concept 'interesting'?"; MOLGEN: "Will the reaction actually take place?") Sometimes a complex test can involve feedback to modify the object being tested (as in META-DENDRAL).

The evidence from our case studies supports the assertion by Newell and Simon that generation-and-test is a law of our science (Newell and Simon, 1976).

Situation⇒action rules

Situation⇒Action rules are used to represent experts' knowledge in all of the case studies. Always the situation part indicates the specific conditions under which the rule is relevant. The action part can be simple (MYCIN: conclude presence of particular organism; DENDRAL: conclude break of particular bond). Or it can be quite complex (MOLGEN: an experiential procedure). The overriding consideration in making design choices is that the rule form chosen be able to represent clearly and directly what the expert wishes to express about the domain. As illustrated, this may necessitate a wide variation in rule syntax and semantics.

From a study of all the projects, a regularity emerges. A salient feature of the Situation⇒Action rule technique for representing experts' knowledge is the modularity of the knowledge base, with the concomitant flexibility to add or change the knowledge easily as the experts' understanding of the domain changes. Here too one must be pragmatic, not doctrinaire. A technique such as this cannot represent modularity of knowledge if that modularity does not exist in the domain. The virtue of this technique is that it serves as a framework for discovering what modularity exists in the domain. Discovery may feed back to cause reformulation of the knowledge toward greater modularity.

Finally, our case studies have shown that strategy knowledge can be captured in rule form. In TEIRESIAS, the metarules capture knowledge of how to deploy domain knowledge; in SU/X, the strategy rules represent the experts' knowledge of "how to analyze" in the domain.

Explanation

Most of the programs, and all of the more recent ones, make available an explanation capability for the user, be he end-user or system developer. Our focus on end-users in applications domains has forced attention to human engineering issues, in particular making the need for the explanation capability imperative.

The Intelligent Agent viewpoint seems to us to demand that the agent be able to explain its activity; else the question arises of who is in control of the agent's activity. The issue is not academic or philosophical. It is an engineering issue that has arisen in medical and military applications of intelligent agents, and will govern future acceptance of AI work in applications areas. And on the philosophical level one might even argue that there is a moral imperative to provide accurate explanations to end-users whose intuitions about our systems are almost nil.

Finally, the explanation capability is needed as part of the concerted attack on the knowledge acquisition problem. Explanation of the reasoning process is central to the interactive transfer of expertise to the knowledge base, and it is our most powerful tool for the debugging of the knowledge base.

ACKNOWLEDGMENT

The work reported herein has received long-term support from the Defense Advanced Research Projects Agency (DAHC 15-73-C-0435). The National Institutes of Health (5R24-RR00612, RR-00785) has supported DENDRAL, META-DENDRAL, and the SUMEX-AIM computer facility on which we compute. The National Science Foundation (MCS 76-11649, DCR 74-23461) has supported research on CRYSALIS and MOLGEN. The Bureau of Health Sciences Research and Evaluation (HS-10544) has supported research on MYCIN. I am grateful to these agencies for their continuing support of our work.

I wish to express my deep admiration and thanks to the faculty, staff and students of the Heuristic Programming Project, and to our collaborators in the various worldly arts, for the creativity and dedication that has made our work exciting and fruitful.

REFERENCES

General

Feigenbaum, E. A. "Artificial Intelligence Research: What is it? What has it achieved? Where is it going?," invited paper, *Symposium on Artificial Intelligence*, Canberra, Australia, 1974.

Feigenbaum, E. A. and J. Feldman, *Computers and Thought*, New York, McGraw-Hill, 1963.

Goldstein, I. and S. Papert, "Artificial Intelligence, Language, and the Study of Knowledge," *Cognitive Science*, Vol. 1, No. 1, 1977.

Gregory, R., "On How so Little Information Controls so Much Behavior," Bionics Research Report No. 1, Machine Intelligence Department, University of Edinburgh, 1968.

Lesser, V. R. and L. D. Erman, "A Retrospective View of the HEARSAY-II Architecture," *Proceedings of the Fifth International Artificial Intelligence-1977*, Massachusetts Institute of Technology, Cambridge, Massachusetts, August 22-25, 1977, Vol. I.

Newell, A. and H. A. Simon, *Human Problem Solving*, Prentice-Hall, 1972.

Newell, A. and H. A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search," *Com ACM*, 19, 3, March, 1976.

DENDRAL and META-DENDRAL

Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg, "On Generality and Problem Solving: a Case Study Using the DENDRAL Program," *Machine Intelligence 6*, Edinburgh Univ. Press, 1971.

Buchanan, B. G., A. M. Duffield, and A. V. Robertson, "An Application of Artificial Intelligence to the Interpretation of Mass Spectra," *Mass Spectrometry Techniques and Applications,* G. W. A. Milne, Ed., John Wiley & Sons, Inc., p. 121, 1971.

Michie, D. and B. G. Buchanan, "Current Status of the Heuristic DENDRAL Program for Applying Artificial Intelligence to the Interpretation of Mass Spectra," *Computers for Spectroscopy,* R. A. G. Carrington, ed., London: Adam Hilger, 1974.

Buchanan, B. G., "Scientific Theory Formation by Computer," Nato Advanced Study Institutes Series, Series E: Applied Science, 14:515, Noordhoff-Leyden, 1976.

Buchanan, B. G., D. H. Smith, W. C. White, R. J. Gritter, E. A. Feigenbaum, J. Lederberg, and C. Djerassi, "Applications of Artificial Intelligence for Chemical Inference XXII. Automatic Rule Formation in Mass Spectrometry by Means of the Meta-DENDRAL Program," *Journal of the ACS,* 98:6168, 1976.

MYCIN

Shortliffe, E. *Computer-based Medical Consultations: MYCIN,* New York, Elsevier, 1976.

Davis, R., B. G. Buchanan, and E. H. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program," *Artificial Intelligence,* 8, 1, February, 1977.

Shortliffe, E. H. and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences,* 23:351, 1975.

TEIRESIAS

Davis, R., "Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases," Memo HPP-76-7, Stanford Computer Science Department, Stanford, CA, 1976.

Davis, R., "Interactive Transfer of Expertise I: Acquisition of New Inference Rules," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence-1977.* Massachusetts Institute of Technology, Cambridge, Massachusetts, August 22-25, 1977.

Davis, R. and B. G. Buchanan, "Meta-Level Knowledge: Overview and Applications," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence-1977.* Massachusetts Institute of Technology, Cambridge, Massachusetts, August 22-25, 1977.

SU/X

Nii, H. P. and E. A. Feigenbaum, "Rule Based Understanding of Signals," *Proceedings of the Conference on Pattern-Directed Inference Systems,* 1978 (forthcoming), also Memo HPP-77-7, Stanford Computer Science Department, Stanford, CA, 1977.

AM

Lenat, D., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," Memo HPP-76-8, Stanford Computer Science Department, Stanford, CA, 1976.

MOLGEN

Martin, N., P. Friedland, J. King, and M. Stefik, "Knowledge Base Management for Experiment Planning in Molecular Genetics," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence-1977.* Massachusetts Institute of Technology, Cambridge, Massachusetts, August 22-25, 1977.

Stefik, M., "Inferring DNA Structures from Segmentation Data," *Artificial Intelligence,* 1978 (in press).

CRYSALIS

Engelmore, R. and H. P. Nii, "A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data," Memo HPP-77-2, Department of Computer Science, Stanford, CA, 1977.

The ubiquity of discovery*

by DOUGLAS B. LENAT
Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

Much of the behavior which we regard as "intelligent" involves some sort of *discovery* process.** This is obvious for some of the most creative and intellectually difficult human activities (identifying an unknown chemical compound, composing a new sonnet, deriving a new cosmological model, ligating plasmids, conjecturing a new theorem, probing the internal structure of a proton, solving the N.Y. Times crossword puzzle, . . .). By the end of this talk we'll see that it's no less true for our everyday activities (understanding spoken English, making a joke, driving from CMU to MIT, cutting cheese, playing board games, finding our way about Boston, solving the Pittsburgh Press crossword puzzle, . . .).

Model-building in science

We in the field of Artificial Intelligence (AI) want to understand how it's possible to do such things, to understand the mechanisms of intelligence. I'll propose a theory of how humans are able to carry out such tasks, and then embody that theory in several concrete, testable models.

Scientific models are useful because they (i) *unify* large masses of empirical data (e.g., the periodic table); and they (ii) *predict* new effects, which subsequently can be tested for by conducting experiments (e.g., Young's two-slit interference test for the wave theory of light).

In very "hard" sciences, objective data are available about isolated and relatively simple phenomena. This enables the construction of small yet quite rigorous *predictive* models, using the language of mathematics (e.g., Maxwell's equations). But in the so-called "soft" fields, the phenomena cannot be measured precisely (the role of the family, the "life after life" reports, etc.), or are not so reproducible (the evolution of life, the origins of WWI, etc.), or (as in the case of human intelligence) cannot be easily isolated for study (e.g., the genetic component of IQ). The resulting models are usually only *descriptive*, and may often be presented in everyday prose (e.g., psychological theories of personality).***

The physicist's model (his set of equations) is better than mere prose because it is able to draw on the power of established mathematics† to make *quantitative* predictions.

But planets and atoms behave in much more regular a fashion than do people. Can we build a hard, predictive sort of model for the phenomena of discovery and creativity? Or are we limited to sterile prose discussions about the mysteries of incubation and illumination (e.g., as in References 3 and 4)? Can we draw only metaphorical pictures (e.g., as in the 'hooked atoms' image that Einstein conjured up when pressed to introspect on how he attained his insights;[5] or as in the "snake swallowing its own tail" dream that inspired Kekule to consider that the structure of the benzine molecule might be a *ring*)? The answer, until just twenty years ago, was unfortunately "We can't do any better." When the answer changed, a whole new science, Artificial Intelligence, was born.

Choosing an approach in science

Each science is differentiated from the others not merely by the set of *phenomena* it claims as its object of study, but also by the *approach* it takes (the science's *view* of those phenomena; its *paradigm*,[6] if you will).

So even though we've decided to study the phenomena of human intelligence (creativity, problem solving, concept formation, etc.), we must still choose a *view* of Man.

If we view Man as an actor whose internal thought processes can't be investigated, then we are called "behavioral psychologists," and we study human behavior. If we view Man as a brain, as a piece of hardware built out of neurons, then we're called "biologists" and we study neurophysio-

* This work was supported in part by the National Science Foundation (MCS77-04440), and in part by the Defense Advanced Research Projects Agency (F44620-73-C-0074).

** Much of the rest of "intelligence" involves algorithmic solving of well-structured problems. That topic will not be emphasized in this paper. We take the position that "algorithms known and used by experts" is just a proper subset of "knowledge experts use to reduce search."

*** Attempts to formalize "soft" phenomena do go on continually, but the interpretation of the resultant rigorous mathematics is often a topic of heated debate (a current example is Catastrophe Theory.[1]

† The power to solve analysis problems in closed form (e.g., to solve a differential equation simply by repeatedly manipulating it according to known transformations), or the power to make approximations when necessary, or the power to somehow "run" the model (to "grind out" solutions to his equations).

SOME HEURISTICS ARE ALMOST ALWAYS RELEVANT

(a) >If X is often true, try to find out exactly where it does and doesn't hold.
(b) >If you must do some new, complicated task, try to arrange things so that the tools, subtasks, etc. are very familiar.
>Look at the extreme cases of the known relationships.
>Ignore minor details until a basic plan is formed.

.
.
.

SOME HEURISTICS ARE RELEVANT WHEN DOING MATH OR SCIENCE

>Even if a theory predicts something strongly, if a very easy test is available, do it (you may be surprised).
>The more important a step is (in a proof, an experiment, etc.), the more stringent verification to which it should be subjected.
>Design experiments so that a negative result will still be interesting.
>When preparing an article's bibliography, cite all possible referees.

. . .

SOME HEURISTICS ARE RELEVANT WHEN DOING BIOLOGY

(a) >It's interesting to study whether a mechanism from one species also operates in another species.
(b) >When choosing which type of experimental animal to use, favor a species about which much is already known.

. . .

SOME HEURISTICS ARE RELEVANT WHEN DOING MOLECULAR GENETICS

(a) >It's interesting to study whether gene control signals from one species are operative in another species.
(b) >E. Coli is a favorite because much is known about its genetics; many of its plasmids have been characterized and are available.
>If you want to introduce DNA between strains of bacteria, then typical vectors to use are plasmids and lysogenic viruses.
>To check whether a gene transferred to a new host has been modified, reintroduce it into its donor.

. . .

SOME HEURISTICS ARE RELEVANT WHEN DOING MATHEMATICS

>If $f: A \times A \times \ldots \times A \to B$, and $A \subset B$, then see if in fact $f: A \times A \times \ldots \times A \to A$;
Else try to find a subset S of A for which $f: S \times S \times \ldots \times S \to S$.
>If a set S has only a few elements, then it's no longer of interest;
But it is worth trying to understand the reason why S is so small.

Figure 1—Hierarchy of heuristic rules

logical responses (e.g., by implanting electrodes). If we view Man as a machine, as an automaton, then we're called "cyberneticists," and we investigate mathematical properties of feedback networks of simple components. If we view Man as a collection of atomic particles, then we're called foolish,** for this is too fine a "granularity" with which to investigate intelligence.

Another view arose about twenty years ago, from three independent sources: (*i*) engineering (Donald Broadbent: Man can be viewed as an information processor with a limited memory system), (*ii*) psycholinguistics (Noam Chomsky: Grammar characterizes human competence; hence, Man can be viewed as a system governed by abstract rules), and (*iii*) computer science and cognitive psychology (Allen Newell and Herbert Simon: Man can be viewed as a processor of symbols). If we adopt that view of Man, then we're working in the field of "Artificial Intelligence."

No one view of Man is "right" or "wrong;" each is adopted because from it we can build a model, which in turn

** Unlike the Brownian motion of atoms in a perfect gas, the fundamental information processes of intelligence are *not random*.

has some practical consequences and uses. When I'm ill, I want to go to a doctor who practices medicine based on Man as a biological organism, not Man as an economic agent. At the present limited state of our understanding, any one view is bound to be simplistic and incomplete.* It is the *whole man* who lives and reacts, even though we can only *view* him, first this way, then that.

The foundation for artificial intelligence

Suppose we view Man as information processor.[7,6] How can we construct models of intelligence that are predictive rather than just descriptive? We might build *operational* models, which can *exhibit* whatever behavior our theory called for. To do this, we need to use a general purpose symbol manipulator: (*i*) a way to specify what processing gets done when, (*ii*) a memory in which to store symbol structures,** and (*iii*) an "engine" that can actually cause such processing to occur. The science of Artificial Intelligence ("AI") sprang into being soon after the invention of one such computational engine: the digital computer. In fact, AI is sometimes called "Machine Intelligence."

The paradigm of AI research

We in AI have evolved the following paradigm:

(i) Choose some human cognitive activity (like playing chess, proving theorems, understanding spoken English).
(ii) Develop hypotheses and eventually a theory about what kinds of information processing could be taking place to produce such ability.
(iii) Incorporate that theory into a computer program, which serves as the model. Make that computer program carry out the original activity, and observe how well it does.†
(iv) Experiment with the program, attempting to find out where the apparent "intelligence" is really coming from.

Over the last twenty years, we've hypothesized and tested scores of models, for several different sophisticated tasks. There have been many successes (such as the Hearsay speech understanding programs), and many failures. Failure often stemmed from *underestimating* how complicated some activity is, which human beings are able to do fairly easily (for instance, driving a car, or understanding someone who's speaking into a microphone.) We once thought that translation by machine could be accomplished by a program which simply looked up each word in a Russian/English dictionary. Legend has it that "*The spirit is willing but the flesh is weak*" was translated into Russian as "*The vodka is fine but the meat is rotten.*"

We've learned a lot from both the successes and the failures. Some of these experiments will be described later; for now, I just want to present a single, very central result.

It turns out that we can model a surprising variety of cognitive activities (recognizing, problem solving, inventing) as a *search* in which the performer is guided by a large collection of informal rules of thumb, which we shall call "heuristics" or "heuristic rules." But what's really exciting is that not only can this single theory (intelligence as heuristic rule guided search) explain the behavior of the brainstorming math researcher and the wandering Boston visitor, but when we go off and build up such models in detail, we find that they can all contain many of the same—or at least similar—heuristics. If we look at many heuristics (see Figure 1), we find that each one seems to have its own sphere of influence, its own domain of applicability, outside of which it is meaningless or useless. There exist many heuristics at all these levels; many of the specific ones are just specializations of one at higher levels (e.g., "*study gene control signals across species boundaries*" is a special case of "*study the scope of biological mechanisms,*" which in turn is a special case of "*study the scope of any phenomenon.*")

HEURISTIC RULE GUIDED SEARCH

The theory

There is a theory of intelligence lurking here:

1. Human cognitive tasks can be cast as *searches,* as explorations wandering toward some goal.
2. We are guided in these searches by a large collection of informal rules of thumb (which we've been calling *heuristics*).
3. We access relevant heuristics in each situation, and follow their advice.

That's it.* It sounds plausible; in fact, it sounds trivial. Yet the models which incorporate this theory are capable of performing many tasks which one would suppose require intelligence (organic chemistry problem solving and research, chess playing, composing musical variations, diagnosing blood infections, discovering new math concepts and conjectures, etc.). (See Figure 2)

* On the other hand, we never capriciously adopt a view with impunity.

** Symbol processing is not the same as mere data processing. See Reference 8. It was some years after its discovery that the electronic digital computer was perceived as a general symbol manipulator.

† AI deviates from cognitive psychology at this stage. Psychologists would run experiments on people, to see if they really do fit the theory. We in AI are more concerned with whether the programs containing the hypothesized mechanisms are capable of *any* sort of "intelligent" behavior—even if it differs somewhat from human performance at that task.

* Notice the conspicuous absence of the word "representation" anywhere in the theory. To design and construct a *model* for this theory would entail grappling with representational issues, much as any running instance of an abstract algorithm must exist on some particular machine. But the validity and power of the theory are independent of representation, just as the validity and complexity of an algorithm are independent of which machine it's implemented on.

Program	Task
AM	Elementary theory formation and discovery in mathematics
DENDRAL	Enumeration of atom-bond graphs of organic molecules
EURISKO	Discovery of new mathematics and new heuristics
Meta-DENDRAL	Abstraction of new fragmentation rules for Dendral
M-METHOD	Improvization of variations on melodies
MOLGEN	Planning of molecular genetics experiments
MYCIN	Clinical diagnosis of blood infections
PECOS	Expert at coding programs in Lisp
PROSPECTOR	Evaluating the mineral potential of sites
PUFF	Medical expert for pulmonary disorders
RITA	Learns how to run programs at remote Arpanet sites
SU/X	Combine many sources' signals into a global model
TEIRESIAS	Expert at extracting new Mycin rules from physicians
UT-ITP	Natural deduction proofs of theorems

Figure 2—Heuristic rule based search program

Intelligence and information

The theory contains two important implicit assumptions, which are worth displaying explicitly:

- Man is viewed as a processor of symbolic information.
- Man exhibits "intelligence" by his performance at various cognitive tasks.

They say that information processing has something to do with intelligence. Let's try to justify that.

Twenty-seven years ago, Alan Turing[9] rejected as meaningless the question "*Can machines think?*" He replaced it with a game, called the Imitation Game (now commonly referred to as the Turing Test). One version of the game would go as follows: A human interrogator is placed in an isolated room. A teletype exists in the room, and by using it he can communicate with a computer and with another human, both located in the next room. The interrogator asks them each some questions, and then must guess which is the human and which is the machine. If we can program a machine in such a way that it fools the interrogator into making the wrong identification at least 50 percent of the time, then we shall say that the machine (as programmed) is "intelligent." Many of you are familiar with this game. Now let me introduce you to a slightly different one:

Thirteen years ago, Keith Gunderson[10] rejected as meaningless the question "*Can rocks think?*" He replaced it with a game, an Imitation Game. A human interrogator is placed in an isolated room. A small hole exists near the bottom of the door, through which the interrogator can shove most of his foot. On the other side of the door are located a human and a rock. Sometimes the human will stomp on the interrogator's foot, and sometimes the rock will fall on it. The interrogator then must guess which one is the human. If we can shape a rock in such a way that it fools the interrogator at least 50 percent of the time, then we shall say that the rock (as shaped) is "intelligent".*

* During the oral presentation of this talk at IJCAI-77, the rock test was demonstrated onstage, with Allen Newell in the role of the unfortunate interrogator.

Why does the dialogue-test sound so much less silly, so much more indicative of intelligence, than the stomping-test? Because unrestricted dialogue is open-ended; it permits genuine interrogation of information and information processing capabilities. The second test doesn't. If you agree that the first test is genuine and the second one bogus, it must be because intelligence has *something* to do with sophisticated processing of massive quantities of information.

Thus it seems that the information processing view of Man is an especially good one from which to study intelligence. Before we see what happens when one tries to build models based on it, let's look at six different kinds of situations requiring intelligence, and see if our theory can handle them all.

Some examples supporting the theory

Everyday problem solving

Suppose we decide to plan a trip from CMU to MIT. How can we find a good route to take? We will probably obtain a detailed road map and begin *searching*. We have some powerful rules of thumb which make our search very short. We look for some main highways that will take us most of the way, and then do some "fixing up" around the termini.

The map-makers aid us by drawing the biggest highways as very thick red lines. Needless to say, ignoring all but the red lines makes the map much simpler. (Thick red lines constrain furiously.) The very general heuristic of *planning* has reduced our search dramatically. One could imagine even a computer being able to solve it. In the next subsection, we'll show that the same kind of analysis can explain episodes of brainstorming (in particular, the inventing some new kitchen gadget).

Everyday invention

Suppose we're confronted with the following problem: we're fond of eating cheese, but it's difficult to use a knife to cut uniform, thin slices. Let's assume that we try

The thinner the knife, the thinner the slices
The softer the cheese, the easier it is cut [except: Brie, . . .]
The slower the slicing, the more uniform the slices
The longer the knife, the more uniform the slices
The rounder the whole cheese, the more irregular slices there will be
etc.

Figure 3—Cheese-cutting relationships

to design an improved tool. We are now facing a search in an enormous space of possibilities. But we have many informal rules of thumb which may help us:

1. Sometimes, there will be a good way (perhaps a recent invention) to do something *more general* than what is strictly asked for.
2. Consider what variables affect the success/failure of the current (inadequate) technique. Look for motivation at the *extreme* cases of the various known relationships involving those variables.
3. Look carefully at what is truly wanted; maybe the problem can be completely bypassed; at least, perhaps it's over-specified.

One of them (#2) says to look for motivation at the *extreme* cases of various known relationships. We know several things about cutting thin slices (see Figure 3). There seems to be some relationship between the thickness of the knife and the thinness of the slices we can cut. So we might consider as an extreme the thinnest knife possible, just a knife *edge*, and perhaps we thereby invent the wire cheese cutter. Or we might look at the extreme case of the relation that says that most cheese can be cut thinner if it's softer, moreso than if it's dry and crumbly. We may ask ourselves if we can get the cheese very soft; completely molten; maybe then pour it into slice-molds, much like making a thin crepe. That's too messy. If we keep the cheese solid, and use a knife, maybe we could just get the cheese *directly under the knife-edge* completely molten; so we've invented the "hot-knife;" General Electric will probably come out with one soon.

This brainstorming seemed like fun. Incorporating heuristic rules into a computer model which could then tirelessly systematically apply them seems like a very promising direction to follow. We'll follow it in a later section. Bear in mind, however, that inventions are rarely made with little or no search. In the cheese-cutting case, there might be hundreds of somewhat relevant heuristics, and many ways of applying each one, and only a few of these combinatorial possibilities are even remotely viable.

On the other hand, both of these other heuristics *can* be used in this situation. The first one is the "inventor's paradox": sometimes the easiest way to solve a problem is to find and solve a more general case, an apparently-harder problem. In the cheese-cutting situation, the heuristic might direct us to look at various devices which "cut": a big scissors? a paper-cutter? a light-saber? the new whirling-nylon-cord grass-cutters? It so happened that recently I came across a cheese slicer which operates remarkably like a paper-cutter. (See Figure 4).

Heuristic number three might have us build a better mousetrap: go into business selling already sliced cheese; or try to devise a "cheese press" that takes the crumbs and squeezes or melts them together into shapes that resemble cheese slices. "Kraft singles" are just such a product: processed cheese shaped into slices and then individually wrapped.

Judging "interestingness"

Let's put our theory to a most severe test, by asking whether it can account for our everyday judgments about

Figure 4—Paper-cutter-like cheese slicer

MOLECULAR GENETICS EXPERIMENT

Transform Thy$^-$ E. Coli into Thy$^+$ E. Coli
by transferring the relevant gene from the bacteriophage Phi-3-T.

EXPANDED PLAN

1. Extract Phi-3-T DNA from bacteriophage Phi-3-T.
   Verify the purity of the DNA by measuring UV absorbtion.
   Check that the molecules are intact, using electron microscope.
   Ensure adequate transforming ability in B. Subtilis.
2. Add restriction enzymes to break the DNA into fragments.
   Completely digest the DNA, by adding enzyme EcoR$_I$.
   Use electrophoresis to estimate the molecular wts. of the fragments.
   Study, hypothesize, and test until confident of what's happening.
   Re-do the digestion, but stopping it after partially complete.
3. Mix the fragments with E. Coli plasmids to construct hybrid plasmids.
   Use EcoR$_I$ to cut out psc101 plasmids from E. Coli.
   After mixing these with the fragments, ligate with T4 ligase.
4. Use these cloned plasmids to transform the E. Coli.
   Grow colonies, selecting for Thy$^+$
5. Verify the Thy$^+$ character of the transformed E. Coli.
   Use a curing technique to remove the hybrid plasmids from E. Coli.
   Confirm the experiment by reintroducing them into cured bacteria and testing that th
   observed frequencies of transform stay constant.

PLAN FOR THE EXPERIMENT

1. Extract Phi-3-T DNA from bacteriophage Phi-3-T.
2. Add restriction enzymes to break the DNA into fragments.
3. Mix the fragments with E. Coli plasmids to construct hybrid plasmids.
4. Use these cloned plasmids to transform the E. Coli.
5. Verify the Thy$^+$ character of the transformed E. Coli.

Figure 5—Molecular genetics experiment

what is and isn't "interesting." This would seem to be the realm of intuition, emotion, taste, and aesthetics—the antithesis of logical empiricism.* If we look, we find that texts on the arts abound with "rules" of composition. They analyze artistic works in terms of symmetry, recency, unconventionality, harmony, utility, etc. A typical rule might say:

> IF two apparently disparate parts of the work are suddenly recognized as being very closely related,
> THEN that increases the interestingness of both parts, and of the whole work as well.

Charles Dickens' popularity is due in no small measure to his mastery of this heuristic of "coincidence:" the reader is always astonished and pleased when two separate characters are discovered to be one and the same person. Escher's art is pervaded by just this sort of co-identification (consider his print where geese and fish "turn into" one another).

* If in fact there is a large non-rational component to such judgments, then it is not surprising if they lie outside the power of any theory founded on a view of Man which ignores that aspect of him. So even a partial success at explaining "taste" in terms of heuristic rules would be significant.

Scientific Problem Solving

Suppose we're designing a molecular genetics experiment, whose ultimate goal is to transfer a gene from one bacteria to another, by using a plasmid (a small, circular DNA molecule). (See Figure 5) We must design a very long chain of steps (perhaps reaching into the thousands), each of the form "raise the temperature to x degrees," "add the following restriction enzyme in order to cut the DNA at a certain place," etc.

It's a common practice to ignore certain kinds of minor steps, and to concentrate on finding a relatively small sequence of important intermediate goals. Afterwards, we can "expand" each subgoal into a detailed list of small steps. (See Figure 5, again) We solved the problem by applying a general heuristic—planning—plus several heuristics specific to molecular genetics.

The heuristic used is just the one we used previously to find a route from CMU to MIT. There, we chose to ignore all minor roads on the map. Here, we chose to ignore the minor experimental steps. This repetition is not accidental; it illustrates the commonality between everyday problem solving and scientific problem solving. Let's see if this sim-

ilarity also occurs between everyday invention and scientific invention.

Scientific invention

Suppose we're confronted with the following problem: we're fond of factoring numbers, but it's difficult to use *division* to cut the numbers up in any uniform, repeatable way. E.g., one time, we may factor 12 into 2×6, another time into 3×4, and another time it crumbles into even more pieces: {1,2,2,3}. We may be content with this situation, or we may try to find out more about factoring, to improve our "cutting tool"—division—or the way in which we wield it.

Such an endeavor would involve searching an enormous space (looking for a new kind of factoring; looking for new facts about factoring). But we have many rules of thumb to help us, including the three rules which were displayed earlier.

One of them (#2) says to look for motivation at the *extreme* cases of various known relationships. In other words, when you've got an interesting function f from A into B, and there are some special subsets of B, extreme kinds of B's, then it's worth defining and studying the inverse images of those special parts of B, i.e., the members of A which map into those special B's. (See Figure 6) The function in this case is "Divisors-of." It maps a number into a set of numbers. (E.g., Divisors-of [12]={1,2,3,4,6,12}.) An extreme case would be when it mapped a number into an extreme kind of set, say an extremely *small* kind of set (e.g., into a singleton, an empty set, a doubleton, etc.).

In other words, this heuristic rule says that it's worth defining and studying the set of numbers with no divisors, with just one divisor, with precisely two divisors, and maybe the set of numbers with precisely three divisors. *Voila'*, we've invented prime numbers. Notice that we used the very same heuristic that led earlier to the invention of the wire cheese cutter and the hot-knife.

Judging "interestingness"

Even allowing that the "heuristic rule guided search" theory could account for the *discovery*—the definition—of prime numbers, could it also explain how a researcher might have noticed that they were valuable and interesting?

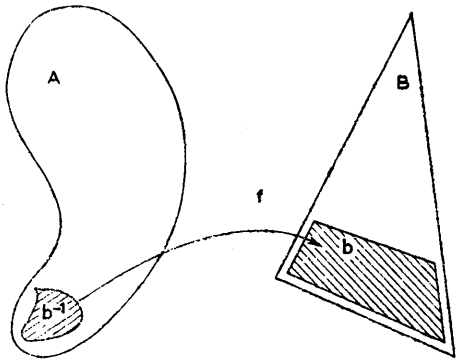
Let's look at how the following four heuristic rules could lead to the conclusion that "Primes" is interesting, soon after it had first been defined.

(#1) IF specializations of concept C have just been created, and the current task is to find examples of each of them,
THEN one method is to look over the known examples of C; they may be examples of some of the new specialized concepts as well.

(#2) IF all examples of a concept C turn out to be examples of another concept D as well, and C was not previously known to be a specialization of D,
THEN conjecture that C is a specialization of D, and raise the "interestingness" value of both concepts.

(#3) IF all examples of a concept turn out to be in the domain of a rarely-applicable function F,
THEN it's worth computing all their F-values (their images under the function F), and studying that collection of F-values as a separate concept.

(#4) IF a concept has very few examples (say, less than 2),
THEN it's worth trying to find out why, but the concept itself is not interesting.



If b is some unusual subset of B,
Then isolate the subset $f^{-1}(b)$ of A.
Figure 6—Go to extremes

The first heuristic tells us that IF we've just created some special kinds of Numbers, and we want to fill in examples of them, THEN one quick way to find such examples is to look at the known examples of Numbers, say the integers from 1 to 1000, and dump each one into whichever new specialized set(s) it belongs. If we do this, we get the following empirical results:

Number with no divisors:	(none found)
Numbers with 1 divisor:	1
Numbers with 2 divisors:	2, 3, 5, 7, 11, 13, 17, 19, . . .
Numbers with 3 divisors:	4, 9, 25, 49, 121, 169, 289, . . .

Heuristic #4 says to rule out any sets which are very tiny; that immediately causes us to lose interest in the first two kinds of numbers, those with zero or one divisor. None of the heuristics have much to say about the next set of numbers (with 2 divisors); what about this final set, numbers which have precisely 3 divisors? Heuristic #2 says, IF all these numbers are also examples of some other special kind of number, THEN they'd be interesting. Well, is there anything special about all of them? Most of them seem to be odd, . . . , *all of them* seem to be perfect squares. That's very unexpected and surprising. All these numbers have integer square roots; that's a very rare thing. Heuristic #3 says that in such a case it's worth actually computing what their square roots are. So let's take their square roots. Lo

and behold! They're precisely the set of numbers with two divisors; i.e., primes. Heuristic #2 explains why we might have noticed this, and indicates that it would cause us to drastically increase the interestingness values of both Numbers-with-3-divisors and Primes. This heuristic was the same one we talked about before, in connection with Dickens and Escher: the rule of startling coincidence.

So we *were* able to fit this kind of judgmental evaluation of "interestingness" into the same theory.

MODELS

The examples I've shown you so far have been merely suggestive. The heuristics which I claimed "explained" various behaviors might really be just comments on those behaviors, ways of rationalizing in hindsight what was actually done. I claim that the heuristic rules can be an *operational* part of the theory, that predictive models can be built which use them for guidance in solving problems and making discoveries. To that end, I'm going to briefly describe a few landmark computer programs that have been written, models based on the theory of intelligence as heuristic rule guided search. These programs form but a thin sliver of the work that has gone on under the label of "Artificial Intelligence."

LT and GPS—general problem solving

Probably one of the earliest AI programs written was the Logic Theorist ("LT", to its friends).[7] It was repeatedly given symbolic logic theorems from *Principia Mathematica,* and its task was to find a formal proof of each theorem. LT had some given axioms and rules of inference. To search for a proof, in a completely exhaustive, systematic manner, could literally be an impossible undertaking! But LT had a few heuristics which constrained its search:

> *". . . selective principles that enable solutions to be found after examining only a relatively tiny subset of the set of possibilities. One such principle—illustrated by the methods in the Logic Theorist—is to generate only elements of the set that are already guaranteed to possess at least some of the properties that define a solution. Another principle—illustrated by the matching process in LT—is to make use of information obtained sequentially in the course of generating possible solutions in order to guide the continuing search. A third principle—illustrated by the use of similarity tests in LT—is to abstract from the detail of the problem expressions and to work in terms of the simpler abstractions."*
>
> —[Newell and Simon 1972], p. 137.[11]

One of LT's heuristics was *constrained generation:* if you want an entity satisfying ten specific properties P1 . . . P10, try to arrange things in such a way that you only have to consider entities already guaranteed to satisfy P1 . . . P7; then just test each one to see if it meets the final three criteria. Another principle was to *learn from your failures:* instead of just blindly backtracking, use the nature of your failure to help guide you in the future, so you won't make exactly the same mistake again.

After they learned the power of heuristics from LT, the same group of researchers then worked on a program called GPS, for General Problem Solver. Its aim was to embed the few general heuristics above in a domain-independent form. It was hoped that *any* problem could be represented in the GPS formalism, and hence solved by GPS. It was the first program to isolate explicitly these general methods:

1. Means ends analysis: Decide what the most important difference is between the current state and the desired goal, locate an operator which is known to be relevant to reducing that kind of difference, and then try to actually apply that operator. With any luck, you'll be one step closer to the goal.
2. Setting up subgoals, equivalent to the idea of *problem reduction,* and to divide and conquer strategies.
3. Planning in an abstraction space.

Two decades of experimenting with GPS have shown that, yes, a large array of problems can be cast in terms that GPS can deal with (states, operators, difference tables), but no, GPS's few general heuristics just aren't powerful enough to solve problems as difficult as can be tackled by humans.

DENDRAL—scientific problem solving

The next giant step along the line of development we're following was taken by Ed Feigenbaum[24] and Joshua Lederberg[22] at Stanford, and independently by Joel Moses[23] here at MIT. They recognized what was lacking: expertise. After all, humans have to train in specialized fields for quite a while before they are able to solve any hard problems in them. This phenomenon might not just be a reflection of human brain frailties, it might indicate a necessary requirement for intelligent problem solving in a complex knowledge-rich domain.

In 1965 they conceived a new kind of computer program, and in the process a whole new approach for AI: they were willing to commit their programs to working on a very specific kind of problem—in Moses' case, symbolic integration; in Lederberg, Feigenbaum, and Buchanan's[25] case, the enumeration of atom-bond graphs of organic molecules (based on analysis of mass spectrograph data and nuclear mass resonance data). For instance, you type in "$C_{10}H_{22}$," and their program types back a list of all 71 structural isomers of decane. They built their program, DENDRAL, around a body of heuristics—not just a few, but a few *dozen* heuristic rules. Some of them were as general as the rules that GPS had, but most of them were specific rules of thumb which they extracted from chemists (e.g., rules for recognizing imbedded subgraphs which are impossible; knowledge of aggregates of radicals which co-occur often; heuristic estimation of the worth of pursuing a particular subproblem;

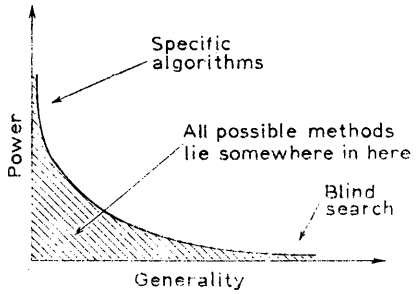


Figure 7—Generality vs. power

rules for recognizing current subproblems which have been worked on before).

The success of this research (measured partly by the large number of articles which have been published in chemical journals) confirmed Feigenbaum's hypothesis about the tradeoff between generality and power. (See Figure 7) Heuristics are distributed along the pictured curve (and below it); the more general they are, the weaker they must be. Experiments with Dendral have demonstrated the importance of using both general heuristics and task-specific heuristics for guidance.

AM—scientific invention

We now jump to some quite recent research by the speaker, which involved collecting *hundreds* of heuristics, to do elementary mathematics research, open-ended discovery. This is scientific *invention,* as compared to Dendral, which performs scientific *problem solving.**

Math discovery as heuristic rule-guided search

The task which this program, called *AM*, performs is the discovery of new mathematics concepts and relationships between them. The simple paradigm it follows for this task is the one specified by our theory:

1. Open-ended math research is viewed as a search, an exploration in a space of partially-developed concepts.**
2. AM is guided in this exploration by a collection of a few hundred heuristic rules. They are rules of varied generality which guide it to define and study the most plausible thing next.
3. In each situation, AM quickly accesses relevant heuristic rules, and then follows (obeys) them.

For example, AM discovered "Multiplication" in four separate ways. This coincidence raised its interestingness value to quite a high level. One of AM's heuristic rules said *"If f is an interesting operation, Then look at its inverse."* This rule was relevant (with f=Multiplication) and was actually followed in this context. It directed AM to define and study the relation "divisors-of." Another heuristic rule which later was obeyed (see Figure 8) said to examine the inverse image of extrema under interesting functions. We've seen this rule before, but Figure 8 indicates more accurately how it appeared in the AM computer program. Just as we saw before, AM was driven by this rule to define prime numbers.

The same heuristic also directed AM to study not only various kinds of numbers having very *few* divisors, but also classes of numbers with very *many* divisors; such "maximally-divisible" numbers were also found to be interesting (by AM, by the author, and by professional mathematicians). In fact, a couple minor new results in number theory were developed regarding them. Still another time, AM was guided by the same heuristic to define and study the pairs of sets whose intersection is empty; i.e., this rule also led to the definition of the relation "Disjointness." This multiple usage of the heuristics is strong evidence that they are truly general.†

Representation of mathematical knowledge

What exactly does it mean for AM to "have the notion of" a concept? It means that AM *represents* that concept somehow in the computer, that there is a data structure of some kind which is meant to correspond to, and contain information about, that concept. While the entire issue of how to represent knowledge has been de-emphasized in this talk, it is helpful to glance at how some typical concepts looked. (See Figure 9)

The representation of a concept is as a collection of *facets,* or slots, (e.g., Definitions, Conjectures, Worth‡) each of which can have some associated value, or entry.

Flow of Control

AM is initially given a collection of 115 core concepts, (see Figure 10) each with only a few facets (slots) filled in. AM repeatedly chooses some facet of some concept, and tries to fill in some entries for that particular slot. Thus a "job" for AM is simply to engage in a mini-research project, to commit a simple act of discovery. Its overall task—to discover interesting concepts and conjectures—is accom-

* This is not the "inductive vs. deductive" issue: both Dendral and author's program (AM) perform quite inductive tasks. The difference is that Dendral is given specific problems to solve (specific mass spectra to identify), whereas AM's activities are open-ended research, with no particular goal specified.

** The goal of this search is ill-defined: to maximize the interestingness value of the concepts which are available for AM to develop further.

† If a supposedly general rule were only involved in one major discovery, there would always be the question of whether the rule is merely a subtle encoding of that discovery.

‡ Unlike the other numbers appearing as part of the pictured 'Primes' concept, the number "800" is merely an estimate, an ad hoc value for the overall worth of this concept. It would be preferable to replace this by a mass of symbolic reasons, by a plausible justification for why this concept was and wasn't useful, interesting, etc. But in lieu of the ability to reason about, create, manipulate, and otherwise use such knowledge, we opt for a simple scalar, a numeric rating which in effect crudely *summarizes* these reasons.

In English and Math notation:

IF f: A → B is an interesting function,
and there's some interesting or extremal subset X of B,
THEN it's worth defining and naming and studying $f^{-1}(X)$.

Roughly as it appeared in the AM program:

```
(IF
  (CURRENT-TASK DEALT-WITH "F")
  AND (F IS-A FUNCTION)
  AND ((WORTH-OF F) IS-GREATER-THAN 600)
  AND (THERE-ARE-KNOWN-ENTRIES-FOR (BOUNDARY-OF (RANGE-OF F)))
THEN
  (FOR-EACH X IN (BOUNDARY-OF (RANGE-OF F))
    (CREATE-A-NEW-CONCEPT
      (WHOSE NAME IS "F-INVERSE-OF-X")
      (WHOSE WORTH · IS (WORTH-OF F)×(WORTH-OF X))
      (WHOSE DEFINITION IS (LAMBDA (z) (F (z) IS-A X)))
      (WHOSE GENERALIZATION IS (DOMAIN-OF F))
      (WHOSE ORIGIN IS "Inverse image of extrema")
      (WHOSE INTEREST IS "(TIES-TO F) (TIES-TO F-INVERSE)"))))
```

Figure 8—Go to extremes, again

plished as a composition of hundreds of these repeated attempts at little discoveries.

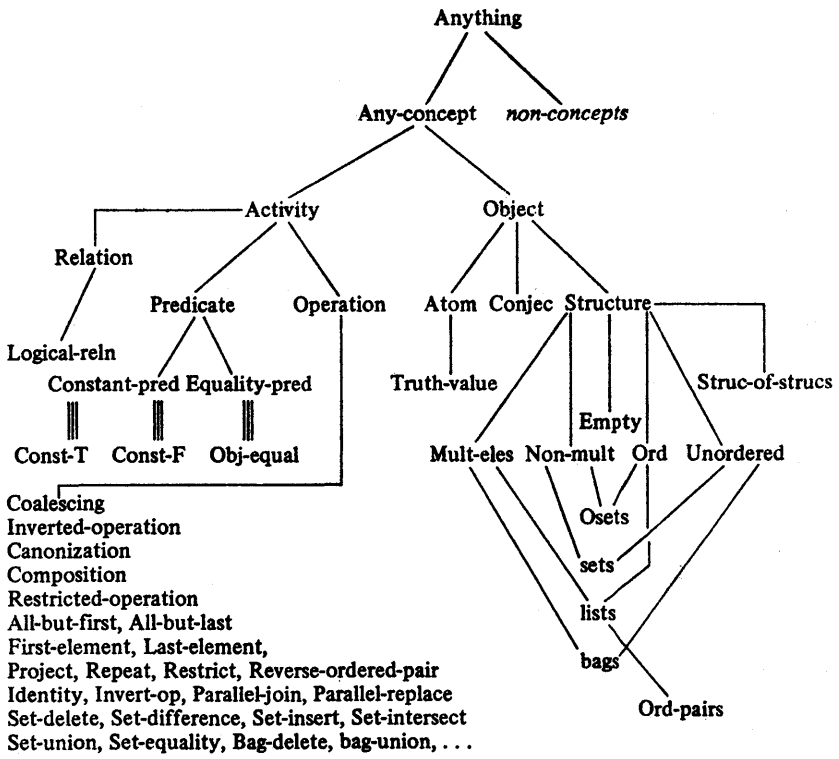
To decide which small job to work on next, AM maintains an *agenda* of jobs, a global queue ordered by priority. (See Figure 11) A typical job is *"Fill-in examples of Primes."* The agenda may contain hundreds of such jobs. AM repeatedly selects the top-rated job from the agenda and tries to find *relevant* rules (i.e., heuristics which, when obeyed, bring AM closer to satisfying that job). Potential relevance is determined *a priori* by where the rule is stored. Each of AM's 250 heuristic rules is tacked onto the most general (or abstract) concept** C to which it's relevant and meaningful. The relevance of these heuristic rules is presumed to be inherited by all C's specializations. A method which can

** It was very important to place each heuristic on as general a concept as possible, to give it a large scope of applicability, to bring it close to the generality versus power curve.

NAME(s): Set, Non-proper Class, Collection, Finite set
DEFINITIONS:
  RECURSIVE: λ(S) [S = {} or Set . Definition
    (Remove(Any-member(S),S))]
  RECURSIVE QUICK: λ(S) [S = {} or Set . Definition (CDR(S))]
  QUICK: λ(S) [Match S with {...}]
SPECIALIZATIONS: Nonempty-set, Set-of-sets, Set-of-numbers
  BOUNDARY: Empty-set, Singleton, Doubleton, Tripleton
GENERALIZATIONS: Unordered-Structure, Collection,
    Structure-with-no-multiple-elements-allowed
IS-A: Kind-of-structure
EXAMPLES:
  TYPICAL: {{}}, {A}, {A,B}, {3}
  BARELY: {}, {A, B, {C, {{{A, C, (3,3,9), ⟨4,{B},A⟩}}}}}
  NOT-QUITE: {A,A}, (), {B,A}
  FOIBLE: ⟨4,1,A,1⟩
CONJEC's: All unordered-structures are sets.
INTUITIONS: Geometric: Venn diagram.
ANALOGIES: {set, set operations} ≡ {list, list operations}
WORTH: 600 [on a scale of 0–1000]
VIEW:
  PREDICATE: λ(P) {x∈Domain(P) | P(x)}
  STRUCTURE: λ(S) Enclose-in-braces
    (Sort(Remove-multiple-elements(S)))
IN-DOMAIN-OF: Union, Intersection, Set-difference, Subset,
    Member, Cartesian-prod, Set-equality
IN-RANGE-OF: Union, Intersect, Set-difference, Satisfying

NAME(s): Prime Numbers, Primes, Numbers-with-2-Divisors
DEFINITIONS:
  ORIGIN: Divisors-of(x) is-a Doubleton
  PRED.-CALCULUS: Prime(x) ≡ (∀z)(z|x → z = 1 XOR z = x)
  ITERATIVE: (for x > 1): For i from 2 to x−1, ¬(i|x)
EXAMPLES: 2, 3, 5, 7, 11, 13, 17
  BOUNDARY: 2, 3
  BOUNDARY-FAILURES: 0, 1
  FAILURES: 12
GENERALIZATIONS: Numbers, Numbers with an even no. of
    divisors, Numbers with a prime no. of divisors
SPECIALIZATIONS: Prime pairs, Prime uniquely-addables
CONJECS: Unique factorization, Goldbach's conjecture
ANALOGIES: Maximally-divisible numbers are converse
    extremes of Divisors-of
INTEREST: Conjec's tying Primes to Times, to Divisors-of, and
    to other closely related operations
WORTH: 800

Figure 9—Two typical math concepts



The diagram above represents the "topmost" concepts which AM had initially, shown connected via Specialization links (\) and Examples links (‖).

Figure 10—Initial concepts

(*Fill in Generalizations facet of "Equality concept"*)
Priority = 700
Reasons = ((No known generalizations of Equality)
(Equality is rarely satisfied)
(Focus of attention: recently worked on Equality))

(*Check Conjecs facet of "Equality" concept*)
Priority = 500
Reasons = ((Worth of Equality has recently risen)
(Conjecs involving Equality have just been found)
(Focus of attention: recently worked on Equality))

(*Fill in Examples facet of "Primes concept"*)
Priority = 400
Reasons = ((No known exs of primes yet))

(*Check Examples facet of "Set-union" concept*)
Priority = 350
Reasons = (Many examples of Set-union recently added)
(Worth of Set-union has recently risen))

(*Check Domain/range facet of "Square-root" concept*)
Priority = 300
Reasons = ((Square-root was unable to return a value for 8)
(Square-root was unable to return a value for 3)
(Should check the domain/range entries for any operation created as an inverse)

(*Fill in new Algorithms for Compose concept*)
Priority = 200
Reasons = ((Empirical: taking too long))

Figure 11—Agenda of jobs

generalize any operation can certainly generalize any composition; any heuristic that knows how to find examples of any concept can find examples of "Numbers-with-3-divisors." And so on. In other words, the aggregate of the Generalization/Specialization relationships among the *concepts* induces a similar graph structure upon the set of *heuristic rules*. (See Figure 12 and also Figure 1, again) This "inheritability property" permits potentially relevant rules to be located efficiently.

The IF-part of each potentially relevant rule is evaluated to determine whether the rule is truly relevant. If so, it's obeyed. At that time, it can direct three kinds of actions or effects:

(i) Facets of some concepts can get filled in (e.g., examples of primes may actually be found and tacked onto the "Examples" facet of the "Primes" concept).
(ii) New concepts may be created (e.g., the concept "numbers which are uniquely representable as the sum of two primes" may be explicitly deemed worth studying).
(iii) New jobs may be added to the agenda (e.g., the current activity may suggest that the following job is worth considering: Find new generalizations of the concept of prime numbers").

Behavior of this Rule System

AM began its investigations with scanty knowledge of one hundred elementary concepts of finite set theory. Most of the obvious set-theoretic concepts and relationships were quickly found (e.g., de Morgan's laws; singletons), but no sophisticated set theory was ever done (e.g., diagonalization). Rather, AM discovered natural numbers, rated them highly, and went off exploring elementary number theory. (See Figure 13) AM discovered scores of well-known concepts (some of them in novel ways,* some of them in several ways**), and a roughly equal number of "losers" (most of which were quickly recognized as such by AM). All this occurred in two cpu hours (see Figure 14).

The AM project has demonstrated that that open-ended scientific theory formation (the defining and exploring of new concepts and relationships) *could* be mechanized, could

* E.g., AM restricted the operation 'Addition' to prime numbers; i.e., it asked "For which primes p,q,r is it true that $p+q=r$?" In such a case, p or q must be "2'," and the other two primes must be a *prime pair*.

** E.g., *multiplication* was discovered in four separate ways: as repeated addition, as an analogue to the Cartesian product of sets, as the cardinality of the union of the powersets of two sets, and as the total number of symbols one gets by replacing (in parallel) each element of bag X by a complete copy of bag Y.

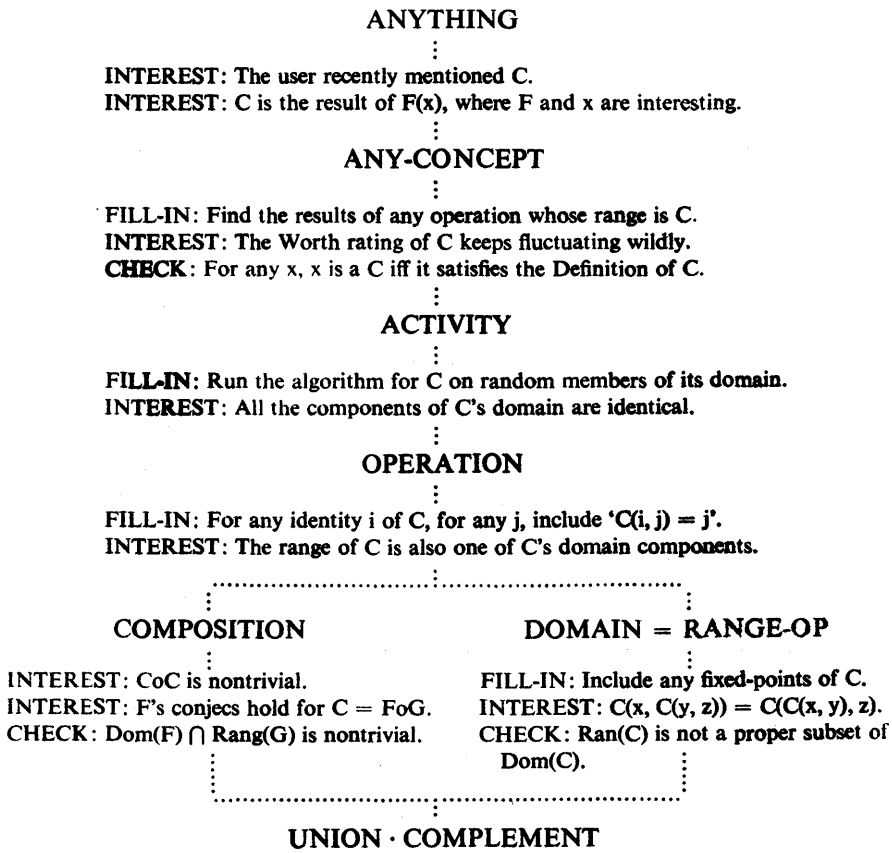


Figure 12—Induced structuring of the heuristics

Sets with less than 2 elements (singletons and empty sets).
Sets with no atomic elements (nests of braces).
Singleton sets.
Bags containing (multiple occurrences of) just one kind of element.
Superset (contains).
Doubleton bags and sets.
Set-membership.
Disjoint bags.
Subset.
Disjoint sets.
Singleton osets.
Same-length (same number of elements).
Same number of left parentheses, plus identical leftmost atoms.
Count (find the number of elements of a given structure).
Numbers (unary representation).
Add.
Minimum.
SUB1 (λ (x) x−1).
Insert x into a given Bag-of-T's (almost ADD1, but not quite).
Subtract (except: if x < y, then the result of x−y will be zero[71]).
Less than or equal to.
Times.
Union of a *bag* of structures.
& (the ampersand represents the creation of several real losers.)
Compose a given operation F with itself (form F.F).
Insert structure S into itself.
Try to delete structure S from itself (a loser).
Double (add 'x' to itself).
Subtract 'x' from itself (as an operation, this is a real zero[72]).
Square (TIMES(x,x)).
Union structure S with itself.
Coalesced-replace2: replace each element s of S by F(s,s).
Coalesced-join2: append together F(s,s), for each member s∈S.
Coa-repeat2: create a new op which takes a struc S, op F, and repeats F(s,t,S) all along
Compose three operations: λ(F,G,H) F.(G.H).
Compose three operations: λ(F,G,H) (F.G).H.
& (lots of losing compositions created, e.g. Self-insert.Set-union.)
ADD^{-1}(x): all ways of representing x as the sum of a bunch of nonzero numbers.
G.H, s.t. H(G(H(x))) is always defined (wherever H is), and G and H are interesting.
Insert.Delete.
Delete.Insert.
Size.ADD^{-1}. (λ (n) The number of ways to partition n)
Cubing
&
Exponentiation.
Halving (in natural numbers only; thus Halving(15)−7).
Even numbers.
Integer square-root.
Perfect squares.
Divisors-of.
Numbers-with-0-divisors.
Numbers-with-1-divisor.
Primes (Numbers-with-2-divisors).
Squares of primes (Numbers-with-3-divisors).
Squares of squares of primes.
Square-roots of primes (a loser).
TIMES^{-1}(x): all ways of representing x as the product of a bunch of numbers (>1).
All ways of representing x as the product of just one number (a trivial notion).
All ways of representing x as the product of primes.
All ways of representing x as the sum of primes.
All ways of representing x as the sum of two primes.
Numbers uniquely representable as the sum of two primes.
Products of squares.
Multiplication by 1.
Multiplication by 0.
Multiplication by 2.
Addition of 0.
Addition of 1.
Addition of 2.
Product of even numbers.
Sum of squares.
Sum of even numbers.
& (losers: various compositions of 3 operations.)
Pairs of perfect squares whose sum is also a perfect square ($x^2+y^2=z^2$).
Prime pairs (p,p+2 are prime).

Figure 13—Concepts discovered

be modelled as heuristic rule guided search, using a few hundred heuristics for guidance. This is a significant verification of the theory of intelligence presented earlier.

AM had some difficulties. As it ran longer and longer, the concepts it defined were further and further from the primitives it began with; while the general set-theoretic heuristics were technically valid for dealing with primes and arithmetic, they were simply too general, too weak to guide effectively. The key deficiency was AM's inability to create and modify new specific heuristics.

Machine: SUMEX, PDP-10, KI-10 uniprocessor
Language: INTERLISP, January 1975 release
Number of small support functions for AM: 200
Number of heuristics: 250
Number of Initial concepts: 115
Number of Final concepts: 300
Average Initial concept: 8 facets
Average Final concept: 11 facets
Average size of a concept: 700 list cells
Average size of a concept: 1 typed page
Total CPU Time: 1 hour
Speedup factor if user advises AM: 3
Total number of jobs executed: 200
Average time allocated per task: 30 secs.
Average time actually used by a task: 18 secs.
Number of winning concepts: 25 (estimated)
Number of acceptable concepts: 100 (est.)
Number of losing concepts: 60 (est.)

Figure 14—Performance statistics

The next step: EURISKO

These findings came to light only because the AM program existed, was run, and was experimented with. Recall that the final step in the AI paradigm is to experiment with the computational model. In this case, what we've learned from AM is guiding our current efforts at writing a program which discovers new task-dependent heuristics as well as new concepts. Our new program, EURISKO, is quite ambitious, because even scientists are very poor at formulating—or even recognizing—new heuristics.

We asked ourselves how new heuristics get discovered; we took the same kind of heuristically-guided approach to the problem that AM would have. By using general rules, we were led to ponder whether heuristics mightn't get discovered and reasoned about in much the same way that math concepts do (i.e., by hindsight, by manipulating existing ones, and so on). We think they might.

So EURISKO's method for discovering and developing

NAME(s): Generalize-rare-predicate
STATEMENT
  English: If a predicate is rarely True, create a generalization of it
  IF-Current-task-deals-with: a predicate P
  AND-IF: P returns "True" less than one tenth of the time
  THEN-Conjecture:
    C1: P is a little less interesting than expected.
    C2: Generalizations of P may be more interesting than P.
  Then-add-a-new-task:
    "Fillin Generalizations of P" because C2
  Then-modify-facet:
    Reduce the Worth of P by 10%, because C1
SPECIALIZATIONS: Generalize-rare-set-predicate
  BOUNDARY: Enlarge-domain-of-predicate
GENERALIZATIONS: Modify-predicate, Generalize-concept
  JUSTIFYING GENERALIZATION: Hill-climbing
IS-A: Heuristic
EXAMPLES:
  GOOD: Generalize "set-equality" into "same-length"
  BAD: Generalize "set-equality" into "same-first-element"
CONJEC's: Special cases of this rule are MUCH more powerful than it is.
ANALOGIES: Weaken-an-overconstrained-problem
WORTH: 600 [on a scale of 0–1000]
VIEW:
  ENLARGE-STRUCTURE: let the structure be {x | P(x)}
ORIGIN: A specialization of "Modify-predicate", via empir. induction
HISTORY
  Summary of examples: 1 good, 1 bad example known; average +0.4
  Summary of conjectures proposed: 3 correct, 1 unclear
  Summary of tasks added to agenda: 2 added, 2 tried, 2 succeeded, +0.8
  Summary of facets modified: 2 Worth lowerings, 1 verified
  CPU Time used: Average 9.4 cpu seconds
  Space used: Average 250 list cells

Figure 15—Heuristic represented as a concept

heuristics is therefore simply to *not distinguish* between concepts and heuristics; i.e., each heuristic is represented internally as a full-fledged concept. (See Figure 15)* Any heuristic which, say, can advise when it's time to generalize any concept, can also automatically tell when it's time to generalize any heuristic. Any method for creating a new concept out of old ones can be used to create new heuristics out of old ones. Evaluating the new heuristics is done just like evaluating any new concepts: by observing them in action, by gathering empirical data about them. Naturally, there are some special heuristics which are meaningful only for reasoning about other heuristics, just as there are some special heuristics that apply only to ligating plasmids.

We've taken the same AM-like approach to solving *most* of the problems that beset AM.

Other heuristic rule guided expert programs

There are several programs like AM in existence, knowledge based expert programs which perform under the guidance of a large collection of heuristic rules.

The MYCIN program[12] contains a couple hundred judgmental rules which were extracted from physicians, and it uses them to make diagnoses of various blood infections.

MOLGEN[13] (still under construction) attacks the molecular genetics experiment-planning problem discussed earlier.

The PROSPECTOR program[14] performs geological analysis of on-site data relayed to the computer by a human expert. It aids him in the evaluation of the mineral potential of exploration sites. Prospector's rules were gleaned from geologists, much as MYCIN's were from physicians.

These programs perform complex *problem solving*. A few programs exist which, like AM, use heuristic rules to guide them in scientific and other *invention* tasks.

META-DENDRAL[15] is a theory formation program which looks over mass spectra and their correct identifications, and then abstracts that data into new fragmentation rules, which are then usable by the Dendral program, as if they had been extracted from a human expert.

The M-Method program[16] for improvising variations on a given melody is based around a body of musicology rules.

Several other such programs are listed in Figure 2.

* Notice its similarity to a typical math concept.

CONCLUSIONS

The main points to remember

- We viewed Man as a processor of symbolic information. Keeping in mind that this is but one narrow view, we found it a useful vantage point from which to study the phenomenon of intelligence.
- Many cognitive tasks can be cast as explorations through more or less enormous* search spaces. "Intelligence" is the ability to zero in effectively on a solution, despite the apparent size of the search space.
- We theorize that humans possess thousands of informal rules of thumb, of varying levels of generality and power, which they use to guide them in these searches. These rules are the prime repository for domain-specific knowledge (i.e., for expertise).
  - —We have seen the relationships among knowledge, search, and intelligence.
    Search is ubiquitous.
    Knowledge reduces search.
    Using knowledge is intelligent.
  - —It is therefore meaningless to debate whether the source of intelligence lies in "search" or in "knowledge:" *both* are prerequisites. They interact synergetically: each would be almost powerless without the other. Search by itself leads to small programs with impossible running times. Knowledge by itself just sits on library shelves and gets dusty.
  - —The theory of intelligence as heuristic rule guided search contains the power of *additivity:* many small pieces of "local knowledge" combine to produce sophisticated "global effects," with a consequent ease of introducing new pieces of knowledge.
  - —This methodology contains the power** afforded by *coarse granularity:* a rule can express any arbitrarily complex heuristic, can embody a meaningful chunk of domain-specific knowledge. In particular, it can have some significance to a human expert. Thus each model of the theory will be self-explaining:[17] it need only keep the user posted of each rule firing. Also, new heuristic rules can be added by a human expert at a level which seems "natural" to him.[18]
- Several example scenarios have been presented, to show that this theory can account for everyday and scientific problem solving, invention, and judging the "interestingness" of newly synthesized inventions. There is a whole lattice of increasingly more specialized heuristics. Some of them, at the very top, are useful in all six types of situations.
- Three kinds of models, based on this theory, were examined (LT/GPS, Dendral, and AM). The efforts are separated in time by periods of about ten years (1957, 1967, 1977) and factors of about ten in the number of heuristics they utilized (3, 30, 300). All three serve to support the theory of intelligence we've been discussing.
- We saw in particular how AM is able to use its heuristics to guide attention (add and modify jobs on its agenda), to synthesize new concepts, to explore them (fill in their facets), and to evaluate their interestingness. This demonstrated that the general heuristic rules are not merely a descriptive commentary, they are an operational part of the theory. AM and its predecessors are not merely lucky accidents.

The goals of AI

Toward a science of science

In the Middle Ages, only a few European artists incorporated linear perspective into their paintings. The techniques were never communicated explicitly; rather, the would-be artist had to serve as an apprentice to a master. Gradually, by example, the student learned the necessary skills. Some time between Giotto and Da Vinci, the artists wrote down, explicitly, the knowledge they were really using, to create perspective. Once they did that, anyone could learn how to create such drawings. In place of mystery was understanding.

There is a similar mystique today, about how doctors can diagnose disease, how mathematicians can discover new theorems, how scientists in general are able to carry out productive research. The doing of science today is not unlike the guilds of the Middle Ages. We scientists rarely try to introspect on our own creativity, to explicate our skills.

One very significant reason for doing Artificial Intelligence research is that it starts the metamorphoses of the nature of scientific fields, from incomprehensible guilds into well-understood technical crafts. It leads to a science of science. AI programs make explicit, in a very usable form, the knowledge necessary to perform some complicated activity, thereby de-mystifying it.

Mental enhancement

A concrete *understanding* of intelligence, as might be achieved by the above "Science of Science", enables attempts to *synthesize* it. What good would a synthetic intellect be?

The ultimate material benefit to be derived from Science is the enhancement of (some aspect of) Man. Medicine views him as an organism, and is able to provide biological enhancement (pharmacological agents and techniques which improve his resistance to disease, his longevity, etc.). Engineering views him as a dynamic actor, and is able to provide physical enhancement (devices and techniques which enable better locomotion than legs, better communication that shouting, etc.). AI views Man as a symbol processor, and provides mental enhancement: programs and techniques

* The less structured the domain, the larger the search space. One extreme of this "tradeoff" is an *algorithm,* which is a highly "compiled" search.

** Promised by general "heuristic search,"[19] and by some rule-based system architectures,[20] but not by "pure" production systems.[21]

which augment his ability to *reason:* automation of routine tasks (bookkeeping, housekeeping*), aids to artists and scientists (partial automation of the "hack" side of composing, proving, diagnosing, exploring, programming), an avenue leading to the understanding (and appreciation of the sophistication) of his own intelligence, and perhaps—in the *far* distant future—even finding his way around the streets of Boston.

ACKNOWLEDGMENT

The author wishes to acknowledge a few of the many individuals who have inspired him with their ideas which have filtered down into this talk: Jon Bentley, Woody Bledsoe, Bruce Buchanan, Edward Feigenbaum, John Gaschnig, Greg Harris, Donald Knuth, Joshua Lederberg, Merle Lenat, John McDermott, Bernard Meltzer, Donald Michie, Allen Newell, Raj Reddy, Earl Sacerdoti, Michael Shamos, Herbert Simon, and Patrick Winston.

REFERENCES

1. Kolata, Gina B., "Catastrophe Theory: The Emperor has No Clothes," *Science,* April 15, 1977, p. 287. See also the spate of letters in the June 15 issue, replying to that article.
2. Poincare', Henri, *The Foundations of Science: Science and Hypothesis, The Value of Science, Science and Method,* The Science Press, N.Y., 1929.
3. Polya, George, *Mathematics and Plausible Reasoning,* Princeton University Press, Princeton, Vol. 1, 1954; Vol. 2, 1954.
5. Hadamard, Jaques, *The Psychology of Invention in the Mathematical Fields,* Dover Publications, N.Y., 1945.
6. Kuhn, Thomas S., *The Structure of Scientific Revolutions,* 2nd. Ed., University of Chicago Press, Chicago, 1970.
7. Newell, Allen, J. Shaw and H. Simon, *Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics,* RAND Corp. Report P-951, March 1957.
8. Newell, Allen, and H. Simon, *Computer Science as Empirical Inquiry: Symbols and Search,* 1975 ACM Turing Award Lecture, CACM 19, Number 3, March 1976, pp. 113-126.
9. Turing, Alan M., "Computing Machinery and Intelligence," *Mind,* Vol. LIX, No. 236, 1950.
10. Gunderson, Keith, "The Imitation Game," in (A. Anderson, ed.) *Minds and Machines,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1964.
11. Newell, Allen, and H. Simon, *Human Problem Solving,* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
12. Shortliffe, Edward H., MYCIN—"A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection," Stanford AI Memo 251, Octorer, 1974.
13. Feitelson, Jerry, and M. Stefik, "A Case Study of the Reasoning in a Genetics Experiment," Heuristic Programming Project Working Paper 77-18, Stanford University Computer Science Department, April 11, 1977.
14. Duda, Richard O., P. Hart, N. Nilsson and G. Sutherland, "Semantic Network Representations in Rule-Based Inference Systems," in (D. A. Waterman and F. Hayes-Roth, eds.) *Pattern-Directed Inference Systems,* Academic Press, 1978.
15. Buchanan, Bruce G., and T. Mitchell, "Model-Directed Learning by Production Rules," in (D. A. Waterman and F. Hayes-Roth, eds.) *Pattern-Directed Inference Systems,* Academic Press, 1978.
16. Zaripov, R. Kh., "Simulation of Functions of Composer and Musicologist on Electronic Computer," *Proceedings of the Fourth IJCAI,* Tbilisi, USSR, 1975.
17. Feigenbaum, Edward A., "The Art of Artificial Intelligence," *Fifth IJCAI,* Cambridge, 1977, pp. 1014-29.
18. Davis, Randall, "Interactive Transfer of Expertise: Acquisition of New Inference Rules," *Proceedings of the Fifth IJCAI (International Joint Conference on Artificial Intelligence),* Cambridge, Mass., August, 1977, pp. 321-8.
19. Nilsson, Nils, *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, N.Y., 1971.
20. Lenat, Douglas B., and G. Harris, "Designing a Rule System That Searches for Scientific Discoveries," in (D. A. Waterman and F. Hayes-Roth, eds.) *Pattern-Directed Inference Systems,* Academic Press, 1978.
21. Newell, Allen, "Production Systems: Models of Control Structures," Computer Science Department Report, Carnegie-Mellon University, Pittsburgh, Pa., 1973.
22. Lederberg, Joshua, Dentral-64: "A System for Computer Construction, Enumeration, and Notation of Organic Molecules as Tree Structures and Cyclic Graphs," NASA Interim Report, 1964.
23. Moses, Joel, *Symbolic Integration,* MIT Report MAC-TR-47, December, 1967.
24. Feigenbaum, Edward A., and J. Feldman (eds.), *Computers and Thought,* McGraw-Hill Book Company, N.Y., 1963.
25. Buchanan, Bruce G., G. Sutherland and E. Feigenbaum, "Heuristic Dendral: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in (Meltzer and Michie, eds.) *Machine Intelligence 4,* American Elsevier Pub., N.Y., 1969, pp. 209-254.
26. Lenat, Douglas B., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," SAIL AIM-286, Artificial Intelligence Laboratory, Stanford University. Jointly issued as Stanford Computer Science Report No. STAN-CS-76-570, and as Stanford Heuristic Programming Porject Report No. HPP-76-8. July, 1976.
27. Lenat, Douglas B., "Automated Theory Formation in Mathematics," *Fifth IJCAI,* Cambridge, 1977, pp. 833-42.

* Controlling the temperature, lighting, security, and other functions around the house.

A panel session—AI in medicine

SESSION CHAIRMAN—SAUL AMAREL *Rutgers University*

Panel Members

Peter Szolovits—Massachusetts Institute of Technology
Harry Pople—University of Pittsburgh
Casimir A. Kulikowski—Rutgers University
Sholom M. Weiss—Rutgers University
J. C. Kunz—The Institutes of Medical Science, Pacific Medical Center
L. M. Fagan—Stanford University
R. J. Fallat—The Institutes of Medical Science, Pacific Medical Center
D. H. McClung—The Institutes of Medical Science, Pacific Medical Center
J. S. Aikins—Stanford University
H. P. Nii—Stanford University
E. A. Feigenbaum—Stanford University
J. J. Osborn—The Institutes of Medical Science, Pacific Medical Center
Mark Stefik—Stanford University
Peter Friedland—Stanford University
R. O. Duda—SRI International
P. E. Hart—SRI International
R. Reboh—SRI International

PANEL OVERVIEW—Saul Amarel

The papers in this session will focus on applications of Artifical Intelligence in the development of knowledge-based systems in medical decision-making situations, in scientific inquiry, and in mineral exploration.

Much of the work which will be reported in this session is being done within the AIM community. [AIM stands for Artificial Intelligence in Medicine.] The AIM project is a national resource sharing activity supported by the biotechnology resources program (BRP) of DRR, NIH, whose principal objective is to promote applications of AI to medicine and the life sciences. The main focus of AIM activity is at the Stanford University SUMEX-AIM project, which provides computer shared resources to the AIM community via national networks. The Rutgers research resource on computers in biomedicine[1] is one of the (BRP-supported) projects in the AIM community. Included in its functions is the organization of AIM workshops.[2] The objective of the workshops is to strengthen scientific interactions within the national AIM community and to disseminate AI-based methodologies, tools, and specific systems that are relevant to AIM. The next AIM workshop (this is the fourth) will be held at Rutgers on June 25-28, 1978. All the participants in this session are expected to attend the Rutgers workshop and to discuss in depth various aspects of their research on knowledge-based systems. The present session will provide an opportunity to bring summary accounts of this research to a wide national audience of computer professionals; a similar session was held last year at the Fifth International Joint Conference on Artificial Intelligence.[3]

Experience with work in AI applications to date shows the following:

(a) The problem of acquiring specific knowledge in a domain, managing it in an AI system, modifying it, and using it appropriately is fundamental. The approach to most designs is incremental and responsive to the fact that the knowledge base in the domain is not stationary. Initially, a relatively low-performance AI system is created to provide the basis for subsequent stages of knowledge acquisition and improvement, which eventually leads to high performance, and expert systems.

(b) Work on applications requires very close collaboration between AI experts and experts in the problem domain. Furthermore, special technical support facilities (e.g., computer networks) can play a significant role in establishing these collaborations and in sustaining their effectiveness. This is a key point which has important implications for organizational and shared resource aspects of applied AI projects.

(c) The development of an expert system within a reasonable time span requires more powerful technologies than those in use today—especially when the knowledge bases will grow from the present 10**2 to 10**3 "facts" to more realistic situations with 10**4 to 10**5 "facts". So far, system development times (from conception to expert level in a research environment) have been 4 to 8 years. To reduce this time span, or to keep it from growing too much as knowledge bases grow, we need more effective methods of knowledge acquisition and organization and more powerful program design environments. Related to this, we need better techniques for interfacing AI programs with experts and users. At a more basic level, we need better schemes for coordinating multiple knowledge bases and for handling inconsistent and/or uncertain information.

Experience of several years in the development of clinical decision-making "expert" systems at MIT to be discussed by Peter Szolovits illustrates the ways in which these issues are encountered in specific AI application projects. Another important conclusion obtained from the MIT experience is that the choice of medical tasks to be approached via AI methodologies/techniques should be made after a careful assessment of the nature and complexity of the task. Only tasks requiring complex knowledge bases and difficult reasoning processes should be approached via AI methods; in many situations simpler computer-based methods may be more appropriate.

One of the key processes in medical reasoning and in scientific inquiry is the interpretation of empirical data in the light of a given body of theoretical knowledge. Much of the work presented in this session is concerned with these processes. Typically, a problem of interpretation involves reasoning about an individual case. Given evidence (data) about the case and a theory within which the evidence is to be understood/explained, find a hypothesis which explains the case on the basis of the theory. While there has been considerable progress in the development of strategies for solving interpretation problems, there still remain several open problems: under what conditions the interpretation process should be controlled by the specific, "low level" features of the case under consideration or by possible "high level" hypotheses and by expectations derived from these hypotheses; how to best represent and keep track of information about a special case, of general knowledge in terms of which the case is interpreted, and of alternative interpretations of the case as the process evolves with time.

Some of these issues will be discussed in depth by Harry Pople in the context of his research (in collaboration with Dr. Jack Myers) on diagnostic problem solving in internal medicine. These are also important issues in (I) the work of John Kunz, Larry Fagan and their collaborators in the domain of pulmonary disorders, (II) the work of Richard Duda, Peter Hart and their collaborators in the domain of mineral explorations, and (III) the work of Sridharan, which is oriented to the development of an AI system for assisting in the formulation of psychological theories of action interpretation.

Processes of planning are becoming another focus of interest in AI applications to natural systems. The synthesis of plans has been a subject of research since the early days of AI. In this session, the synthesis of treatment plans in a medical context (consultation in glaucoma) will be discussed in the paper by Casimir Kulikowski and Sholom Weiss. AI approaches to the development of plans in the context of scientific experimentation (in molecular genetics) will be discussed in the paper by Mark Stefik. Research on the analysis/interpretation of plans is of a more recent vintage in AI and it has been stimulated by AI work in "real life" situations. Work on this problem will be presented in the context of AI applications to scientific work in the papers by Stefik (molecular genetics) and Sridharan (psychology).

Some of the systems presented in this session have already reached high levels of expertise in limited domains. CASNET/glaucoma at Rutgers and INTERNIST at the University of Pittsburgh and PROSPECTOR at SRI are in this category. There are also several "younger" systems that are still in a process of intense development and growth. In some of them techniques developed/tried in previous knowledge-based systems are being assimilated and adapted (e.g., the structure of MYCIN[4] is being used in the PUFF project) and the DENDRAL[5] experience as well as ideas from several more recent AI efforts are being used in the MOLGEN project. On the other hand, in the BELIEVER task discussed by Sridharan, a new system, called AIMDS, is being developed to provide an appropriate environment for theoretical work on the task.

We are now at a point where substantial achievements have been made in AI applications and there are several new efforts that are starting to build on top of past experience. In general, there has been good progress in this area and the prospects for the future of high performance in AI expert systems are bright.

The following is a summary of the papers to be presented at the NCC session.

REFERENCES

1. Amarel, A., "Computer Based Modeling and Interpretation in Medicine and Psychology," in *Computers in Life Science Research,* Siler and Lindberg (eds.), Faseb and Plenum, 1975.
2. Amarel, S., C. A. Kulikowski, N. S. Sridharan, and D. Sridharan, *Proceedings of The First Annual AIM Workshop,* June 14-17, 1975, DCS Rutgers University, New Brunswick, N.J., 1976.
3. Amarel, S., et al, "Reports of Panel on Applications of AI," *Proceedings of the Fifth International Conference on Artificial Intelligence,* MIT, Cambridge, Mass., August 1977.
4. Shortliffe, E. H., *Computer-based Medical Consultations: MYCIN,* American Elsevier, New York, 1976.
5. Buchanan, B. G., G. L. Sutherland, and E. A. Feigenbaum, "Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in *Machine Intelligence 4,* Meltzer and Michie (eds.), American Elsevier, New York, 1969.

THE DEVELOPMENT OF CLINICAL EXPERTISE IN THE COMPUTER—THE EVOLUTION OF CLINICAL DECISION-MAKING PROGRAMS AT MIT—Peter Szolovits

We at the Clinical Decision Making Group at MIT's Laboratory for Computer Science, along with our collaborators at the Tufts-New England Medical Center Hospital, have been engaged for several years in the development of computer programs which embody expert medical knowledge. We have applied techniques developed in the study of Artificial Intelligence (AI) to encode expert clinicians' approaches to handling a number of important medical problems: (1) prescribing appropriate doses of digitalis glycosides to patients with congestive heart failure or arrhythmias (the DIG program), (2) taking the history of the present illness

of a patient with kidney disease (the program is called PIP), and (3) diagnosing and treating patients with acid/base and electrolyte disturbances. We have also studied the application of decision analytic techniques to the testing and treatment of Hodgkin's disease, the differential diagnosis of acute renal failure, the value of coronary surgery to the individual patient, and other problems.

Throughout many of these projects, we have discovered that our initial estimate of the difficulty of the problem has been far from what we finally decided. Naturally, we have often underestimated the magnitude of the effort needed to collect data, generate plausible decision structures, and choose and build computer implementations for the data base and reasoning components of the program. Not so obviously, however, we have also found that some problems succumb to much simpler techniques than what we had originally estimated to be needed.

The key to these observations, we now think, concerns the degree to which limited human-like expertise in a computer program is adequate to cover the great majority of situations which arise in the program's domain. To the extent that a simple statistical model or a small set of decision rules or protocols can be made to work, any program which uses those models will be straightforward. It is only when the task involves complex reasoning to deal with a wide variety of medical possibilities, when a "deep" understanding of a problem is needed, or when the medical situation can consist of a complex combination of many simultaneous difficulties that the programs we write (and the experts' knowledge which with models) must be sophisticated.

The presentation will review a number of attempts to build programs for digitalis therapy, to contrast them in terms of the magnitude of medical knowledge involved in each and the corresponding sophistication demanded of the computer implementation.

REFERENCES

1. Pauker, S. G., G. A. Gorry, J. P. Kassirer, and W. B. Schwartz, "Towards the Simulation of Clinical Cognition: Taking the Present Illness by Computer," *American Journal of Medicine*, 60, 1976, pp. 981-996.
2. Silverman, H., *A Digitalis Therapy Advisor*, LCS Report, MAC TR-143, 1975, MIT, Boston, Massachusetts.
3. Szolovits, P. and S. G. Pauker, "Research on A Medical Consultation System for Taking the Present Illness," *Proceedings of the 3rd Illinois Conference on Medical Information Systems*, University of Illinois, Chicago Circle Medical Center, Urbana-Champaign, Illinois, November 1976.

THE ROLE OF HYPOTHETICAL REASONING IN DIAGNOSTIC PROBLEM SOLVING—Harry Pople

INTERNIST is a computer-based system designed to deal with complex diagnostic problems in internal medicine. An important aspect of this task domain is the need to synthesize ad hoc diagnostic categories to characterize patients with two or more disease processes at one time. The presentation will review the role of hypothetical reasoning in the INTERNIST approach to this concept-formation component of the diagnostic task.

REFERENCES

1. Pople, H. E., "The Synthesis of Composite Hypotheses in Diagnostic Problem Solving; An Exercise in Hypothetical Reasoning," *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, August 1977, MIT, Cambridge, Massachusetts.
2. Pople, H. E., "Artificial Intelligence Approaches to Computer-based Medical Consultation," *Proceedings of the IEEE Intercon.*, New York, 1975.
3. Pople, H. E., J. D. Myers, and R. A. Miller, "The DIALOG Model of Diagnostic Logic and Its Use in Internal Medicine," *Proceedings of the International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975.

STRATEGIES OF GLAUCOMA TREATMENT PLANNING—Casimir A. Kulikowski and Sholom M. Weiss

The ultimate goal of most difficult medical consultations is to produce advice that will help formulate a plan of treatment with the greatest likelihood of success for the patient. Recently, several computer consultation programs using Artificial Intelligence techniques have developed different strategies for treatment planning.[1-3] The strategies used in the CASNET/Glaucoma program.[2,4] are designed to select plans for the long-term management of complex glaucoma cases over many follow-up visits.

Although many subjective rules of clinical judgment enter into the formulation of treatment plans, the expected outcomes are usually based on an understanding of the physiological effects of the treatment. In CASNET/Glaucoma, a causal model describes the mechanisms of disease as well as their expected changes following treatment. A two-level data structure is used: general treatment plans subsume ordered sequences of specific treatments. The initial step is to select a general plan of treatment on the basis of the patient's diagnostic categorization. The choice of specific treatment within the plan is determined by many of the individual characteristics of the patient, through the use of scoring functions for the enhancement and inhibition of selections. For example, the status of the fellow eye of a patient becomes a crucial factor when surgery is contemplated: the potential risk may be too great if there is little or no vision in the fellow eye.

For follow-up visits, the effectiveness of any current treatments must be judged and compared with the expected benefits of the alternatives suggested by the selection strategy. The diagnostic status of the patient must also be reassessed, in the case that changes dictate a complete change of treatment plan. In the event that the patient remains within the scope of the original general plan, but the original specific treatment is not sufficient to control the progression of disease, the program selects the next, stronger medication in

the sequence. Potential side-effects or complications are monitored and may result in changing the order in the sequence of specific therapies.

The CASNET/Glaucoma program's recommendations were compared to those of a panel of experts at the 1976 meeting of the American Academy of Ophthalmology and Otolaryngology.[5] The resulting discussion is serving as the basis for testing a variety of new model-based strategies for treatment.

REFERENCES

1. Shortliffe, E. H., S. G. Axline, B. G. Buchanan, T. C. Herigan, and S. N. Cohen, "An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy," *Computer Biomedical Research,* 6, p. 544, 1973.
2. Weiss, S., *A System for Model-based Computer-aided Diagnosis and Therapy,* Ph.D. Dissertation, Rutgers University, 1974.
3. Silverman, H., *A Digitalis Therapy Advisor,* MAC TR-143, Massachusetts Institute of Technology, January 1975.
4. Weiss, S., C. A. Kulikowski, and A. Safir, "Glaucoma Consultation by Computer," *Computer Biol. Med.,* 8:25, January 1978.
5. Lichter, P. R. and D. R. Anderson, *Discussions on Glaucoma,* Grune and Stratton, New York, 1977.

USE OF ARTIFICIAL INTELLIGENCE FOR INTERPRETATION OF PHYSIOLOGICAL MEASUREMENTS: PULMONARY FUNCTION DIAGNOSIS AND I.C.U. VENTILATOR MANAGEMENT—J. C. Kunz, L. M. Fagan, R. J. Fallat, D. H. McClung, J. S. Aikins, H. P. Nii, E. A. Feigenbaum and J. J. Osborn

Interpretation of physiological measurement data is a central process in medical diagnosis. Biomedical engineering has succeeded in providing a wide variety of quantitative measures of physiological state, but the biomedical community has made only limited effort at systematically attempting to automate the interpretation of the medical significance of these quantitative measurements.

We characterize the problem of medical data interpretation by its medical and its sociological aspects. We choose to highlight the following features of the medical problem:

Judgmental: Deductive and inductive reasoning are both used to conclude a diagnosis given a medical situation;
Uncertain: There is only a loose relation between measured result, physiological state, overall clinical state, and effects of therapy;
Redundant: Measurements have redundancy with each other;
Context-dependent: Interpretation depends upon the clinical situation;
Time-dependent: The value and meaning of information changes.

In addition, we highlight the following features of the sociological situation in which medicine is practiced:

Experience: Good clinical practice is characterized by experience, in addition to didactic knowledge;
Interpretation: The consultant, either human or automated, should explain the process followed in making a consulting diagnosis.
Expectation: Patient condition is viewed as improving or deteriorating in relation to an expectation of disease course.

We will describe a general architecure for a medical interpretation system which recognizes these features of the problem. We have chosen Artificial Intelligence (AI) techniques for our approach to medical data interpretation. AI offers an approach to symbolic reasoning which is amenable to automation. The symbolic manipulation allows us to represent the knowledge of the clinical problem in a direct form, i.e., as structured relations among medical states and measurements. We report here on our early progress with use of AI to address two medical problems as we see them.

The PUFF system is now in routine use in our hospital for interpreting pulmonary function test measurements. PUFF interprets the medical significance and implications of input quantitative test data and patient history. Using a production rule formalism, PUFF has a set of about 60 physiological rules of the general form "IF condition THEN draw conclusion." The rule formalism has allowed our pulmonary physiologist to define the medical interpretation problem in his terms, using a level of judgment and a compensation for the uncertainty and redundancy of the data which he feels appropriate. The system is implemented in a version of the MYCIN system on the PDP 10 and in a straightforward way on the PDP-11. The system operates in a batch mode, accepting input from the data collection computer and printing out interpretations. In a 144-case evaluation, the system showed 86-93 percent agreement with the physiologist in diagnosing the presence of major pulmonary disease. For comparison, the physiologist changed his initial diagnosis of the presence and severity of major pulmonary disease upon further consideration in 42 of 107 cases during an early validation process.

The VM system is designed to interpret the clinical significance of quantitative measurements from our ICU patient monitoring system. Every 2-10 minutes, VM will accept the 40 measurements comprising the monitoring data and make a context and time dependent interpretation of the significance of data. The system first compares measured data with expected values in order to identify the possible presence of alarm conditions. The system has rules which define criteria for accepting data as physiologically indicative or possibly spurious. Next the system focuses its attention on the measurements which categorize the clinical state of the patient, and then it attempts to identify possibly advantageous therapies. VM now includes the capability to process rules describing optimal procedures for removing patients from mechanical respirators. The program will print suggestions to clinicians concerning optimal control of respirator settings and choice of specific respirator types. Detection of unexpected mechanical failures and potentially significant

changes in the patient's physiological measurements will be interpreted and reported to the clinician.

MACHINE INFERENCE FOR MOLECULAR GENETICS: METHODS AND APPLICATIONS —Mark Stefik and Peter Friedland

An increasing amount of research in Artificial Intelligence is involved with difficult applications which require representing and using significant amounts of specialized knowledge in a computer. Several researchers[2,4] have suggested that this represents a shift of paradigm in AI—from seeking performance via powerful general methods toward a recognition that many difficult processes are knowledge intensive. This view emphasizes the use of large amounts of specialized knowledge in a fashion that facilitates their effective use and interaction.

The MOLGEN project at Stanford is an example of this trend in research. MOLGEN the joint effort of computer scientists and geneticists at Stanford, is broadly concerned with the application of symbolic reasoning to molecular genetics. Molecular genetics is a vigorous and rapidly developing discipline with many logical and medium-sized problems for potential computer application. The main thrust of the MOLGEN effort has been in the development of programs to assist in the design of laboratory experiments. Experiment design requires information about problem solving goals, available laboratory techniques, and strategies for combining these techniques to achieve the desired goals. Work in computer-assisted design has been concentrated in a limited class of analysis experiments and a limited class of synthesis experiments.

Development of performance programs in the MOLGEN project has required work on several fronts. Experiment planning involves representation of many kinds of laboratory methods and objects. To avoid creating separate packages for each kind of knowledge, we have developed a schema-based representation package[5] with some fairly general capabilities. This part of our work, which occupied much of our first year of research, was considerably influenced by recent work in representation languages.[1] At the same time, the process of planning experiments and resolving conflicts has drawn on recent work in problem solving.[6] Prior to beginning our work in this area, we did a substantial review of problem solving methods.[7]

Projects like MOLGEN foster the art of applying the principles and tools of AI research to bear on specific problems. This usually involves adapting and combining these tools; it always involves comparing and selecting among alternative approaches. One of the motivations for these efforts is their utility as case studies in methodology. We believe that artificial intelligence will mature as a discipline only by confrontation of its basic methods with the challenges of various applications. From the case studies are slowly emerging the bits of a theory, which consists of an increasingly well characterized collection of paradigms for solving problems and representing knowledge. One example of the results of a case study from our own work is in Reference 8.

REFERENCES

1. Bobrow, D. G. and T. Winograd, "An Overview of KRL, A Knowledge Representation Language," *Cognitive Science,* Vol. 1, No. 1, January 1977, pp. 3-46.
2. Feigenbaum, E. A., "The Art of Artificial Intelligence: I. Themes and case studies of knowledge engineering," *Proceedings of the Fifth International Conference on Artificial Intelligence,* 1977, pp. 1014-1029.
3. Feitelson, J. and M. J. Stefik, "A Case Study of the Reasoning in a Genetics Experiment," *Heuristic Programming Project Report* HPP-77-18 Working Paper, May 1977.
4. Goldstein, I. and S. Papert, "Atrificial Intelligence, Language, and the Study of Knowledge," *Cognitive Science,* Vol. 1, No. 1, January 1977, pp. 84-123.
5. Martin, N., P. Friedland, J. King, and M. J. Stefik, "Knowledge Base Management for Experiment Planning in Molecular Genetics," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* August 1977, pp. 882-887.
6. Sacerdoti, E. D., "The Nonlinear Nature of Plans," *Proceedings of the Fourth International Conference on Artificial Intelligence,* September 1975, pp. 206-214.
7. Stefik, M. J. and N. Martin, A Review of Knowledge Based Problem Solving As a Basis for a Genetics Experiment Designing System, Stanford Computer Science Department Report STAN-CS-77-596, March 1977.
8. Stefik, M. J., Inferring DNA Structures from Segmentation Data: A Case Study, Heuristic Programing Project Report 78-3, January 1978.

COMPUTER-BASED CONSULTANT FOR MINERAL EXPLORATION—R. O. Duda, P. E. Hart, and R. Reboh

For the past two years, we have been developing an interactive system of computer programs called PROSPECTOR that is intended to act as a consultant to field geologists on problems of mineral exploration. The performance of the system is based on an internal set of models of various types of mineral deposits, and on a taxonomy of rocks, minerals, and other geological concepts. The models and taxonomy, termed the knowledge base, are provided to the system over a period of time by a panel of expert geological consultants. The field geologist using the system furnishes a set of observations, often incomplete and uncertain, about the particular prospect under investigation. The principal tasks of PROSPECTOR are to compare the observations of the user against the models, to find the most promising matches, to identify for the user the most important missing observational data, and finally to form an estimate of the likelihood that a deposit in fact occurs on the prospect. Viewed abstractly, PROSPECTOR attempts to aggregate the knowledge of expert geologists in order to apply that knowledge to particular mineral exploration problems.

The models of mineral deposits are represented in a special data structure termed an inference network. A node in the network represents an arbitrary geological situation like "The prospect lies within 200 miles of a subduction zone" or "The ratio of pyrite to bornite on the prospect is less

than 1." Each node has an associated probability. A subset of nodes are distinguished as model nodes; that is, a node in this subset represents the situation that a mineral deposit satisfying a model occurs on the prospect.

Nodes in the network are connected by directed arcs. An arc corresponds to either a logical connective or to a probabilistic inference relating the nodes it joins. An important form of reasoning in PROSPECTOR is to propagate probabilities along arcs from one node to the next. The computations for propagation are based on subjective probabilities furnished by the experts, and are described in Reference 1.

The inference network formulation is particularly well-suited to our purpose because it allows models to be built up incrementally over time by defining new situations and relations. Each relation, together with the nodes it joins, is called a rule, and for this reason PROSPECTOR is termed a rule-based inference system.

The situations described by the nodes in the inference network are frequently compounded out of simpler situations; in the previous illustration the occurrence of pyrite and bornite are evidently the primitive geological situations of interest. It is often important to reason about these primitive constituents of a more complex situation. For example, the user may have already supplied the system with the observation that no sulfide minerals occur on the prospect. Since both pyrite and bornite are sulfides, PROSPECTOR should be able to infer that it is meaningless to inquire about their ratio. To support reasoning such as this, we represent each situation in the inference network in a more detailed structure called a semantic network.[2,3] The semantic network for the previous example would explicitly articulate a situation in which both pyrite and bornite are present and are in the prescribed ratio. Furthermore, additional arcs denoting the subset relation would link pyrite and bornite to the mineral taxonomy in general and to the sulfide minerals in particular.

PROSPECTOR operates as a mixed-initiative system. At any time the user may take the initiative and supply the system with observational data or may request further information of several sorts. When the user relinquishes initiative, PROSPECTOR asks the user to establish by observation the likelihood that a particular geological situation occurs on the prospect. PROSPECTOR selects situations—that is, nodes in the inference network—that will have the most pronounced effect on the probabilities of the model nodes. Thus, PROSPECTOR in effect pursues a strategy of trying to confirm or refute hypotheses about the occurrence of a mineral deposit on the prospect.

REFERENCES

1. Duda, R. O., P. E. Hart, and N. J. Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," National Computer Conf., *AFIPS Conf. Proc.*, Vol. 45, 1976, pp. 1075-1082.
2. Hendrix, G. G., "Expanding the Utility of Semantic Networks Through Partitioning."
3. Duda, R. O., et al, "Semantic Network Representations for Rule-Based Inference Systems," to appear in *Pattern Directed Inference Systems*, D. A. Waterman and R. Hayes-Roth (eds.), Academic Press, 1978 (in press).
4. Hart, P. E. and R. O. Duda, "PROSPECTOR—A Computer-Based Consultant System for Mineral Exploration," to appear in *Joint International Association for Mathematical Geology*, Vol. 10, No. 5-6 (in press). Also appears as Tech Note 155 AI Center, SRI International, Menlo Park, CA.

A dependency-based modelling mechanism for problem solving

by PHIL LONDON
University of Maryland
College Park, Maryland

INTRODUCTION

The design and implementation of automatic, general purpose problem solvers is a complex task. Although there are several possible methodologies in which to couch a theory of general purpose problem solving, a popular approach in recent AI research has been *problem reduction*. Problem reduction is a technique in which the problem solver's goals are broken down into sets of (presumably) simpler subgoals where the problem solver's task of discovering a solution for each is easier than discovering a solution for the main goal.

Implicit in this assumption of simplicity of subgoal solutions is the requirement that each subgoal be treated independently. Otherwise, the advantage gained by specifying simpler subgoals is defeated by the requirement for continuous interaction among those subgoals' solutions. Thus, an important aim of the research in general problem solving has been to discover schemes whereby autonomy of the subgoal solution process can be preserved.

There are times, however, when interaction among subgoal solutions is inescapable, either because of the order in which the subgoals are specified or because of innate interdependence of subgoal solutions. A promising approach to handling subgoal interaction is to allow the subgoal processes to assume that they are independent of each other and to provide a meta-process which watches the progress of the subgoal solutions. The "watcher" can assure that the currently active subgoal process does nothing inimical to the accomplishments of other completed subgoal processes.

There are several strategies that have been investigated to deal with the interaction of subgoals. I will concern myself in this discussion to the class of solutions that treats the subgoals sequentially, allowing each subgoal process to make use of results of solutions generated for previous subgoals. This approach is typified by the systems of Rieger and London,[1] Tate,[2] Waldinger,[3] and Warren.[4] Several other approaches to subgoal interaction in problem reduction exist. These include what could be called the *parallel* approach (Sacerdoti[5]) in which subgoals are solved independently and then subgoal solutions are merged together, eliminating redundant operations. A third approach could be called the *hybrid* approach (Dawson and Siklossy[6]), a combination of sequential and parallel approaches, in which the plan is expanded by choosing that subgoal to work on next which least interferes with the already existing solution. Finally, there is the so-called *linear* solution of Sussman,[7] which, like the parallel, assumes total subgoal independence at plan time, and corrects for interaction problems at execution time.

In the sequential type of solution, a powerful mechanism for maintaining a world model is required wherein each successive subgoal can determine the effects on the environment which have resulted from the sequence of actions thus far proposed by prior subgoal processes. Each subgoal process is autonomous in that it does not care what particular solution was generated by previous subgoal processes. Nevertheless, it should be aware of the effects those solutions had on the environment. The modelling mechanism must be adequate to represent these alterations to the environment.

Within this setting, I will discuss a modelling technique called *Dependency-Based* (DB) modelling which relies on an explicit representation of justifications for beliefs held by the problem solver. Using these justifications, the DB modelling mechanism is able to determine the relevant lines of inference to pursue during problem solving. Then, I will discuss three particular problem solving difficulties and I will propose remedies which will revolve about the DB modelling technique. The three difficulties are:

1. Subgoal violation detection—the ability to detect that a subgoal whose existence is still required by the main goal has been undone by the solution to a brother subgoal
2. Descriptor binding—the ability to recognize the correct binding in the plan for certain objects which are known in advance only by descriptions of their features
3. Maintaining a consistent world model—when subgoal violations are detected, the ability to relieve those interactions by modifying the sequence in which actions are applied. Demands are made on the modelling mechanism to update efficiently the world model to reflect these changes.

In following sections, I will motivate these problems and

discuss a proposed solution founded on a modelling mechanism based on dependency nets.

BACKGROUND

Maintaining world models for problem solvers is a task which has received much attention. Much of this effort was stimulated by discussion of the "Frame Problem" by McCarthy and Hayes.[16] The Frame Problem points out the difficulty of representing the influence of actions on a problem solving environment.

The modelling mechanism I will describe is based on a representation called *dependency nets*. A dependency net is a graph whose nodes represent beliefs and whose arcs represent logical justifications for those beliefs. Utilization of explicit dependency representations for world modelling is due to Fikes.[8] Dependencies have also been used to represent problem solving assumptions to help control processes such as backtracking.[9]

The power of dependency nets as the basis for modelling mechanisms is derived from their ability to direct the inference processes which draw conclusions from new beliefs about the world. Dependency nets provide a convenient structure for maintaining knowledge about derived relations among assertions. In this way, changes in belief can be efficiently accommodated into the model by allowing already derived relations to direct the propagation of information through the network. Duda, et al.[10] have investigated the propagation of information through networks of relations, although their primary focus was to deal with contributions of justifications with variable confidences to the overall strength of belief of a hypothesis. My model draws heavily on the notion of propagation of information through the network, though these capabilities have been enhanced to deal with problem solving difficulties they did not consider.

A detailed investigation of the nature of dependency structures has been performed by Doyle.[11] While his primary focus was on control issues (e.g., dependency-directed backtracking[9]), mine is on modelling issues. Among the important ideas he developed are the notions of well-founded support and non-monotonic deduction. Well-founded support is justification of a belief which ultimately relies only on fully believed premises. This notion is exercised fully in the theory described in this paper. I have not dealt with his concept of non-monotonicity which states that the justification for belief of some assertions may be based on the non-justifiability of their negations or of certain other assertions. There are many fundamental differences between our systems, though, because of the differences in purpose (control vs. modelling).

Besides dependencies, there are several popular techniques for modelling the state of the world during problem solving. Among these are context-layered databases[14,17] and "plan structure" models.[3,5] They both have advantages in terms of efficiencies for specific operations which are described later. They are both deficient, though, when compared to a dependency-based model in that they do not have the ability to record already derived relations among assertions and to utilize those relations at appropriate times.

THE DEPENDENCY-BASED MODELLING MECHANISM

As described in the Introduction, there are difficulties inherent to the sequential problem reduction approach which arise when subgoals are not independent. Their remedies will be addressed within the DB modelling mechanism for the Commonsense Algorithm (CSA) problem solver.[1]

Dependency nets

The DB modelling mechanism relies on a representation known as a *dependency net*. A dependency net is a network whose nodes are assertions representing beliefs held by the problem solver. The arcs of the network represent relations among the assertions believed by the system. In this way, not only are the system's current beliefs represented, but so are the justifications for holding those beliefs.

The dependency network representation provides the capability to make explicit the interrelationships among facts and beliefs held by the problem solving system. The network's power is derived from its ability to guide inference and deductive procedures when new facts are added to the network or, more importantly, when the truth value of one of the system's beliefs changes. For example, assume the system believes a fact P1 to be true. Suppose then, that an attempt to deduce the believed truth of P2 occurs (e.g., the problem solver wants to know if the goal P2 is already true). The deductive mechanism determines that P2 should be believed FALSE and furthermore, the reason is that a deductive rule was retrieved that stated if P1 is true then P2 is false.* These facts are explicitly represented in the network notation of Figure 1. The truth values required for the relationship between P1 and P2 to be valid (the *schematic truth values*) are denoted on the arc. The believed truth values for P1 and P2 are denoted inside the nodes (e.g., P1:T).

A great deal of flexibility results from the explicit representation of dependency relations. For example, if some event occurs which causes the truth value of P2 to change to TRUE, the system knows it can immediately draw the inference that P1 is false.** If support for P1 is explicitly represented, then those supporting beliefs can be updated in a similar way. In this manner, an alteration in truth value propagates through the network from consequents to antecedents by a process I will call *antecedent propagation*.

Let us consider a slightly more complicated example of antecedent propagation. Suppose, in determining the truth value of an assertion P, the deductive component returns that P should be considered FALSE because P1 and P2 are

* In the context of this discussion, *deduction* will be defined as the process of determining the believed truth value of a specified assertion.

** I take *inference* to be that process which computes the consequences of the assertion of a belief or the change in truth value of a belief.

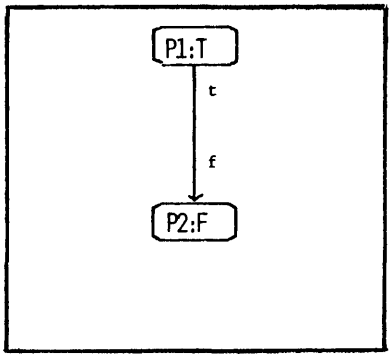


Figure 1

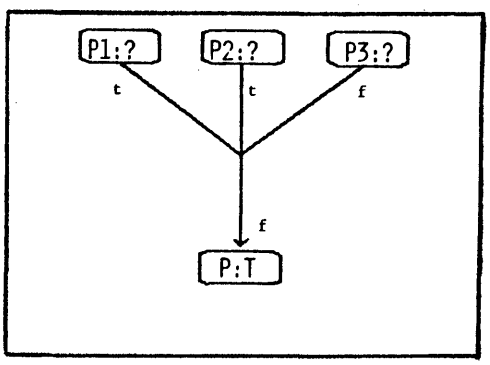


Figure 3

believed TRUE and P3 is believed FALSE. This relation is represented in Figure 2. In this case, if P should change truth value, it is known that at least one of P1, P2, P3 must also change truth value. Deduction could be applied to each to determine which had changed, and this truth value alteration could be propagated backward through the net as before. In order that the model always be consistent, whenever the consequent of a dependency relation changes truth value, deduction must be applied to each of its antecedents.

A great savings in computation could be achieved if the application of deduction to some of the antecedents of an altered consequent is avoided. A reasonable approach is to mark each of the antecedents P1, P2, and P3 as having uncertain truth value (Figure 3), and let this uncertainty propagate through the net. In this way, applications of deduction can be postponed until they are deemed relevant. If uncertainty is ever propagated to an assertion which is considered important, then deduction can be applied at that point and the effect of the result of this deduction can be propagated through the net. What makes an assertion important in a problem solving environment is described later.

Dependency net behavior when an assertion's truth value is altered is not governed solely by antecedent propagation. Truth value alteration can be propagated from antecedents toward consequents (*consequent propagation*), though this is logically different from antecedent propagation. For example, in Figure 2, if P1 changes truth value then little can be said about the truth value of P. The support of belief P has been lost, but it may be that alternate support for the currently believed truth value of P can be derived. When support for an assertion's believed truth value is lost, its truth value becomes uncertain. To determine if there was an alteration of truth value as a result of the lost support, deduction must be reapplied to an assertion whose truth value might not have changed at all. Therefore, all means of support for an assertion should be represented in the net.

For any assertion P whose truth value is desired, the deductive component should not merely retrieve the first applicable rule discovered to provide support for P's deduced truth value, but *all* applicable rules. I will call the application of all valid rules, *exhaustive support*. Although exhaustive support places a heavy demand on the deductive component, it is crucial for some applications to which the dependency net will be applied during problem solving. Having exhaustive support applied to all assertions in the network guarantees the well-founded support which helps avoid the dangers of circular justifications (see Reference 11).

Multiple applicable deductive rules occur frequently and are represented as in Figure 4. In this example, the falsity of P is independently supported by the truth of PA1 and PA2 and by the truth of PB1 and the falsity of PB2 and PB3. If some event should cause PB1 to be believed FALSE, the belief in the falsity of P does not change because of the still valid support provided by PA1 and PA2.

If, as in Figure 2, a fact P has only singular support, then a change in truth value of an antecedent (say P1) signals a change in truth value of the consequent P. The actual support for the new truth value of P must be determined by deduction, but one can hypothesize on logical grounds that P1 has something to do with it.

As with antecedent propagation, the modification of truth

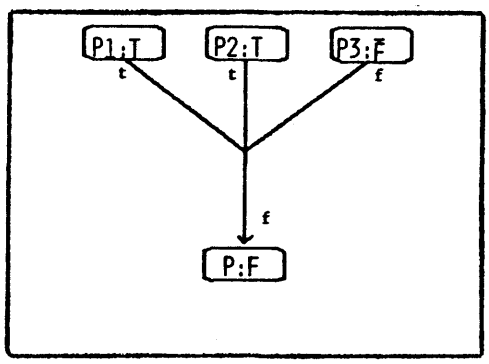


Figure 2

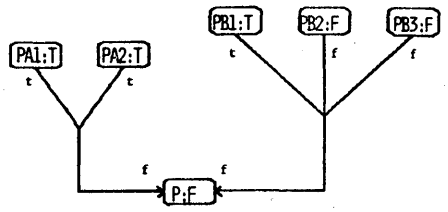


Figure 4

values of beliefs in the dependency net can be propagated along dependency links, though in the direction of the consequent by consequent propagation. Rather than applying needless deduction, assertions whose truth values have become uncertain can be so marked with uncertain truth value until a solid determination of its truth value and new support relations is required.*

A brief system overview

The dependency-based model is intended to serve as a component of a complete problem solving system whose functional modules are depicted in Figure 5. The DB model can be considered as an intelligent interface lying between the planning and deductive components. The planning component frequently requires truth values for various assertions (e.g., goals) and those truth values are determined by the deductive component. The DB model serves as an intermediary who remembers the results of deductions and who applies a highly constrained inference process for the purpose of maintaining a consistent set of beliefs about the problem solving environment. The only facts contained in the DB model are facts which are relevant to the problem solving environment because the model is activated only by assertions passed to it by the planning component. Additionally, a desirable feature of this organization is that the DB modelling component can be studied independently of any particular planning module.

Deduction rules

Dependency relations represented in the network arise from two sources. The first of these sources is the set of assumptions made by the problem solver itself. Such assumptions include the expected consequences of performing actions, selection of objects to participate in plans, and action sequence relations (that the results of an action not only depend on the action but also the context in which the action is performed).

The second source of dependency relations is the deductive mechanism. Dependencies which arise as a result of the invocation of deduction represent logical relationships among beliefs. The schematic examples in the previous section are instances of logical dependencies.

The representation of logical support rules in the dependency-based modelling scheme is rather straightforward. All deduction rules simply are represented as dependency net schemata. For instance, the dependency relation represented in Figure 6 was generated from the deduction schema in Figure 7.

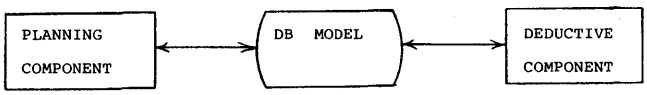


Figure 5

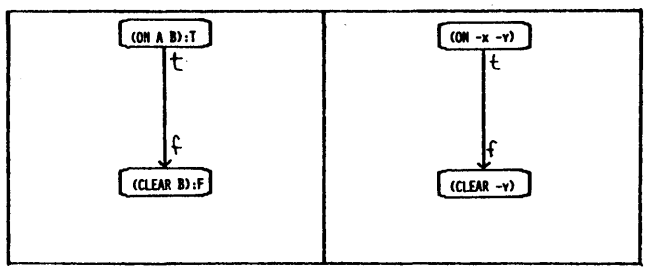


Figure 6 Figure 7

For a logical relation between assertions to be valid, the assertions must be assigned the schematic truth values. The schematic truth values of the support *rule* in Figure 8 represent that if P(x,y) and Q(y) are true then R(x) is false. The instantiation of this rule in Figure 9 illustrates an instance in which the relation is valid; the antecedents represent *actual support* for the consequent. Figure 10 shows an instance of this rule which still describes the same logical relationship among the predicates P, Q, and R (called *potential support*), though the rule is not currently valid since the truth values of the assertions do not match the schematic truth values required by the relationship.

An example of dependency net behavior

The goal of the application of logical support rules is to construct and maintain a richly interconnected network of assertions such that for any assertion P, every other assertion contained in the net which bears a logical relationship to P will be connected to P along some path in the network. These paths connecting logically dependent assertions will be discoverable by antecedent or consequent propagation. In order to maintain a consistent model, all valid alternative means of support as represented in the rules matching the derived truth value for a given assertion must be retrieved. For this retrieval, the particular method used by the deductive search is not significant to the DB model. The intent of this modelling method is to limit the amount of deduction performed during inference while maintaining a consistent model.

Example

Let us now consider an example which describes how the dependency network behaves in order to maintain model consistency. Then, in the next section, we will see how this behavior can be "fine-tuned" to limit the number of deductions to just those that are required.

Consider the set of rules in Figure 11. Suppose that these rules are sufficient to maintain a consistent model in some domain. Suppose also that in this domain, the problem solver possesses a strategy for achieving (making true) the goal predicate P3. This strategy is depicted in Figure 12. The

* Consequent propagation of uncertain truth values is similar to the truth maintenance technique utilized by Fikes.[8]

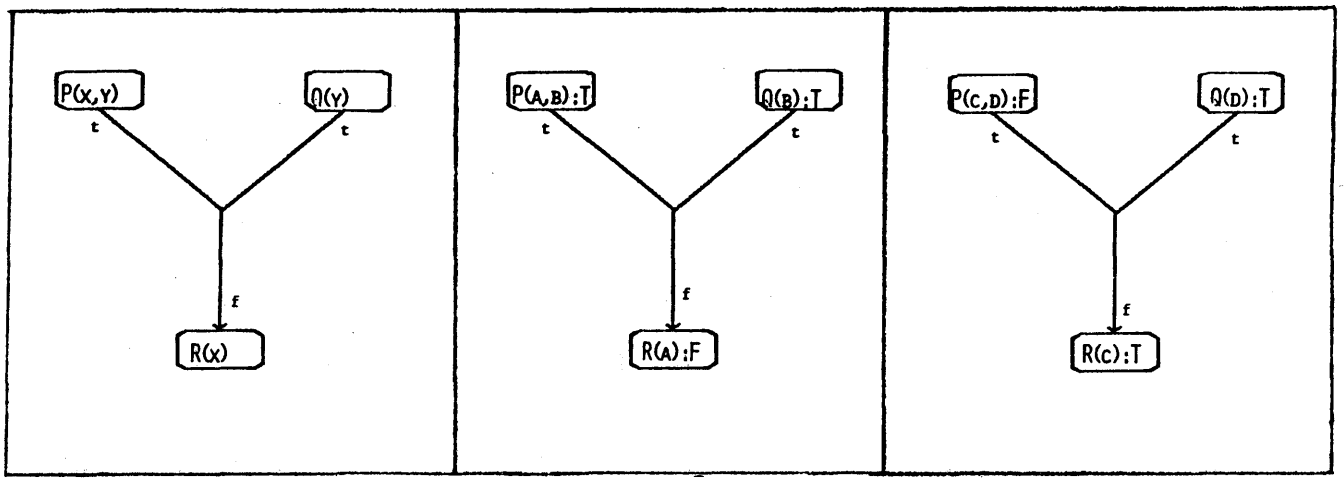


Figure 8 Figure 9 Figure 10

strategy is encoded in the Commonsense Algorithm representation.[1,12] In this case the strategy pattern states that performing the action A causes the predicate P3 to become true (or the goal P3 to be achieved) provided at the time the action is performed that P5 is true.

Now assume that in the initial world (the problem statement) P1, P4, P5, and P10 are true (Figure 13). The problem solver attempts to determine if goal P3 is already true. If so, it need do nothing further. The truth value of P3 is deduced, a value of FALSE is returned, and the strategy of Figure 12 is applied in an attempt to achieve it.

The assertion P3 having truth value FALSE is entered into the existing dependency net and exhaustive support for this belief is determined and drawn into the net (Figure 14). In this case, RULE1 and RULE2 are retrieved since their consequents match P3 and they assert the falsity of P3. The general retrieval algorithm being applied assures that an instantiation of a support rule will be knitted into the net if the consequent and all antecedents which are already in the net match the schematic truth values required of them.

Following the same line, the truth value of P2 should be deduced to determine whether the potential support provided by RULE2 indeed represents actual support for P3:F.

Suppose a truth value of TRUE is deduced for P2. The appropriate support rules for P2:T are retrieved and this process continues until the truth value of all assertions are known and exhaustive support is explicitly represented in the net. This dependency net, which shows P3 and all valid lines of support for belief in P3's falsity, is displayed in Figure 15.

The problem solver now applies itself recursively to P5, the subgoal specified by the strategy. Deduction of the truth value of P5 is not required since it is already explicitly represented in the dependency net. Furthermore, no search for support is required since every assertion in the net is already exhaustively supported. Since the subgoal is solved, the action A can cause the truth of P3, our original goal. This is explicitly represented in Figure 16 by an *action-consequent* link.

Inference behavior spontaneously occurs as the network reacts to this alteration in truth value of one of its assertions. Network inference amounts to an attempt to maintain consistency of the world model. By antecedent propagation, the inference process concludes that P4 must have become false (RULE3 from Figure 11 states this relationship between P3 and P4 explicitly). Similarly, it is concluded that at least one of P1, P2 must have become false. The model update so far is depicted in Figure 17. Let us assume that when deduction is applied to P1 and P2, it is concluded that only P2 has

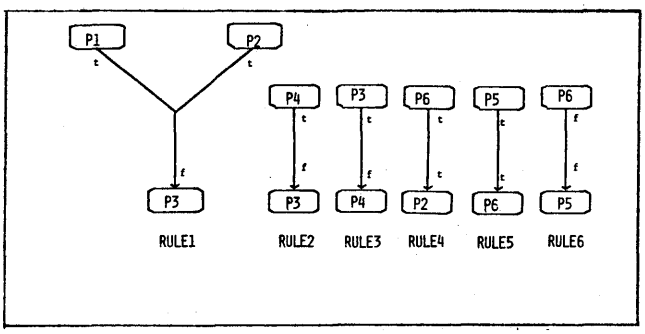


Figure 11

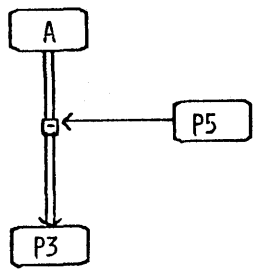


Figure 12

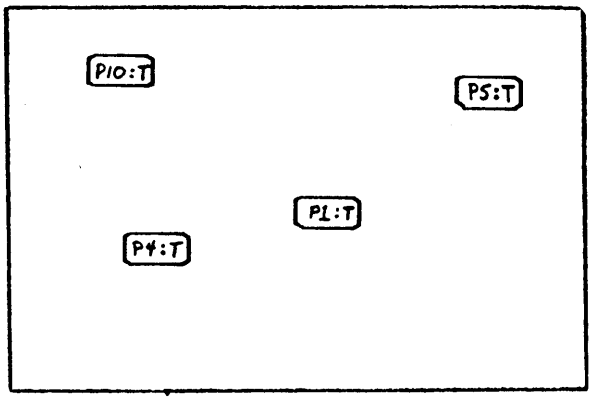


Figure 13

become false. It follows, if P2 is false then so are P6 and P5.

The propagation of alteration of truth values continues until the condition of the network stabilizes. The model is then consistent and the effects of performing the action A will have been accommodated. Every assertion in the net has an associated truth value which can at this point be considered to reflect the current state of the world as modified by the action A. Exhaustive support is required in order to assure that the effects of an alteration of truth value are propagated to every assertion which bears any logical relationship to the assertions which are immediately represented as being altered by performance of the action. In practice, the ability to maintain exhaustive support is limited by the capabilities of the deduction mechanism and the time allotted to search for lines of support.

Application of rules

If it were possible to be more selective about when to apply the deductive mechanism to determine whether a rule retrieved as potential support indeed represents actual support, one could greatly reduce the overhead which would be required to maintain exhaustive support in large dependency

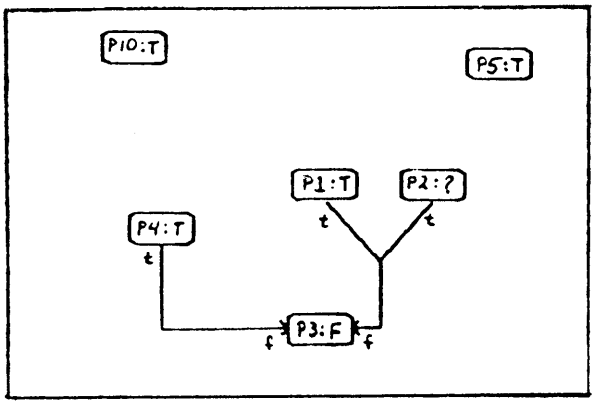


Figure 14

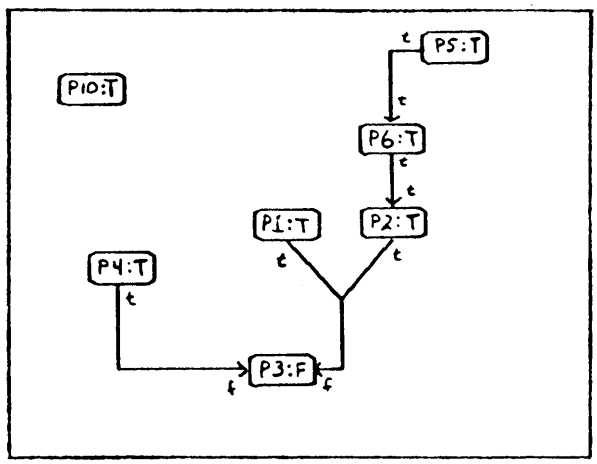


Figure 15

nets. In this section, I will describe modifications to the algorithm for verifying actual support which limit the amount of deduction required to maintain exhaustive support. The method for retrieving exhaustive support locates all support rules which provide potential support for the truth value determined by the deductive component for the consequent assertion. Those rules which do not provide actual support should not be weeded out until the need for actual support arises. This would occur, for instance, if the assertion changes truth value.

Consider the situation in Figure 14. The truth value of P2 is required to determine if RULE2 represents an actual support relation for P3:F. Recalling that the very purpose of exhaustive support is for drawing inferences from an alteration of truth value of an assertion in the net, we note that if P3 were never to change from false to true, there might never be a need to determine if RULE2 is indeed an actual support rule. The decision on the truth value of P2 can therefore be delayed until P3 changes truth value. This will become a key source of efficiency in the DB modelling

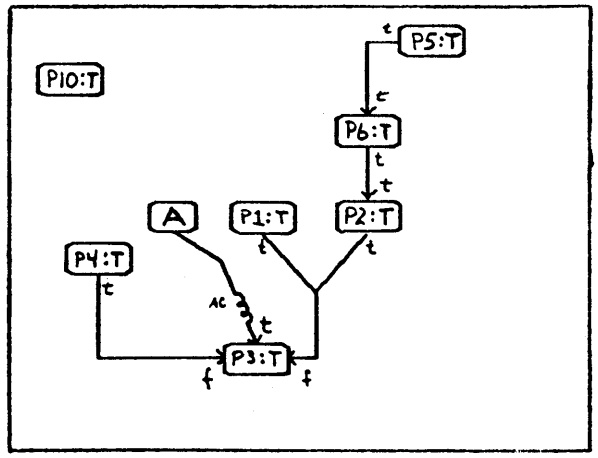


Figure 16

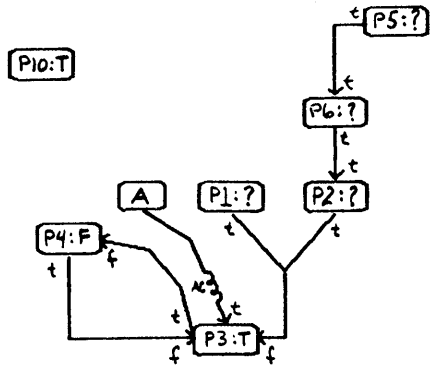


Figure 17

scheme, the efficiency being derived from the ability to focus the deductive process on only those assertions currently considered crucial to maintaining model consistency. In other words, *potential* support need only be verified when *actual* support is required.

To illustrate this point, let us introduce three new rules to the set depicted in Figure 11. The new rule set appears in Figure 18. With this rule set, when RULE2 was applied to support P3:F, exhaustive support for P1:T would have demanded the deduction of truth values for P8 and P9. This would have been a clearly wasteful operation since the truth value of P1 was never altered. In addition, since P8 and P9 were not already in the net the problem solver is not interested in them anyway.*

To realize this idea, the modifications to the information propagation algorithms are as follows.

(1) Exhaustive Support—Knit into the dependency net all support rules whose consequent matches the assertion and its deduced truth value and whose antecedents *which are already in the net* match the schematic truth values of the rule. Mark any newly generated assertions (antecedents which were not already in the net) with '*', meaning that deduction has not been pursued on this node so that this rule provides actual support only if this node has proper truth value.

(2) Antecedent Propagation—When an assertion A changes truth value:

  [1] Change truth value of nodes which are the sole antecedents of valid rules of which A is the consequent.

  [2] In rules which have multiple antecedents mark all the antecedents '?' (meaning that the truth values of these assertions might have changed). If any nodes are now marked '?*' then it is necessary to apply deduction to that node with respect to the net before the consequent was changed for the purpose of determining whether the rule was providing actual support for the altered consequent. If the deduced truth value for the antecedent does not match that required by the rule, then the rule can be discarded because it did not represent actual support. If the truth value of the antecedent matches the rule, then mark the antecedent as having uncertain truth value '?' and put it on the antecedent propagation queue.

These modifications minimize the number of deductions required by inference in accommodating the alteration in belief of an assertion. This inference process determines all relevant effects of a modification in belief of an assertion by the problem solver. Deduction is only performed when required and the network representation is flexible enough to make bookkeeping for the process a fairly simple matter.

* It should be noted that intermediate assertions like P2 and P6 in Figure 16 only serve the purpose of tying "important" assertions together. Important assertions are those stated in the initial world and those for which truth values are explicitly requested by the problem solver (i.e., goals).

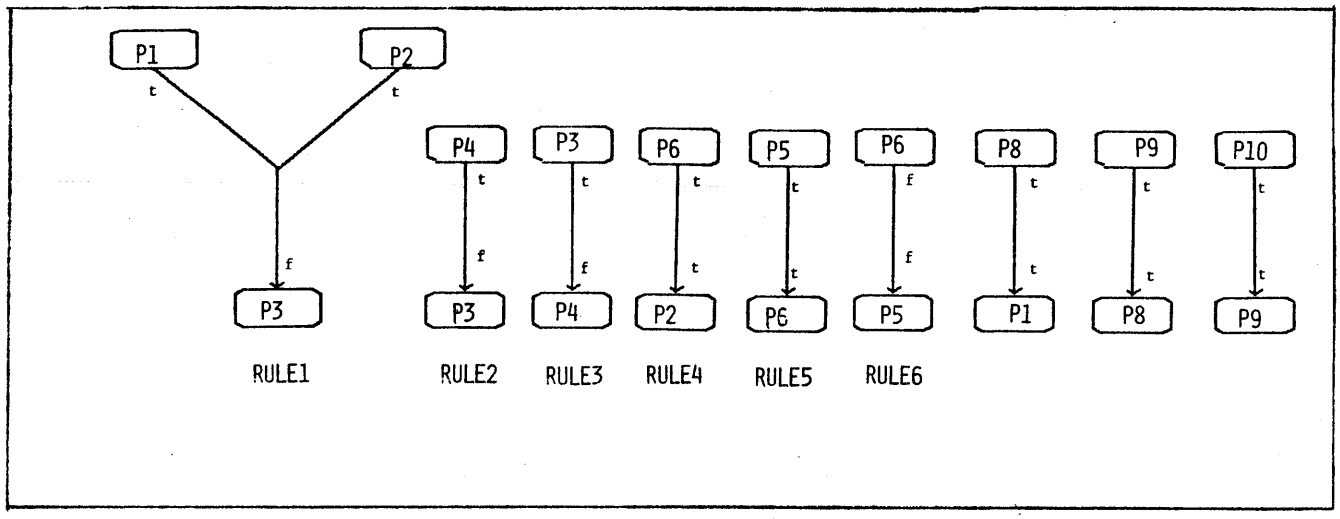


Figure 18

A comment about deduction

My purpose here has not been to propose a new mechanical theorem prover; I have intentionally ignored the problem of deduction entirely. Rather, my purpose has been to demonstrate that the goal structure of a plan and a representation for the justifications of beliefs can be used to guide an efficient modelling technique. Deductions are performed only in response to requests for truth by the problem solver and in determining the actual support relations for believed assertions. One strength of this approach lies in its ability to avoid deduction for the maintenance of a consistent world model until the truth of assertions in that model have a bearing on the outcome of the planning process.

There is an extensive and well-known body of literature on the subject of mechanical deduction.[13] The dependency-based modelling mechanism places no restrictions on the structure of the deductive process. The deductive component could be implemented as a mechanical theorem prover or as a search through the collection of support rules provided. The DB model merely assumes the existence of some deductive capability which can determine the truth value of an assertion with respect to the context established by other known beliefs and that the determined truth value is compatible with that which would be derived if the support rules were applied by some search process for the purpose of deduction.

APPLICATION OF THE DEPENDENCY NET MODEL

In the Introduction, I described three difficulties for general problem solvers that synthesize solutions to conjunctive subgoals by considering the subgoals sequentially. These difficulties arise when the subgoals are not independent. I will now outline solutions to these difficulties within the DB model framework.

Subgoal violation detection

If techniques are to be devised to deal with potential interaction among solutions to sets of conjunctive subgoals, then some method for detecting those interactions must be available. Remembering that one of our aims is to maintain as much autonomy as possible for the solution process of each subgoal, we require a scheme for subgoal violation detection which can run in the background without affecting the activity of the problem solver directly. Subgoals must remain intact until the main goal for which the subgoal has been defined has been achieved. A popular scheme for guaranteeing the temporal scope of a subgoal has been to use protection.[1,3,7]

The violation detection scheme described here utilizes inference as a restricted search process for a general system of violation detection. This violation detection scheme derives naturally from the inferences which occur to accommodate a change in belief in the net. To illustrate, assume a goal G has been solved by the problem solver. A protection violation would occur if, after G is asserted to be true (after it is solved), a consequence of that assertion is to make a protected assertion false. It follows that detecting protection violations can be accomplished by hypothesizing G to be true before actually making it so, allowing the dependency net to react to that assumption, then determining whether a change of truth value propagated to an assertion marked protected. The violation could be eliminated by processes such as described by Rieger and London,[1] Tate,[2] or Waldinger.[3]

It should be noted that this type of processing could conceivably be performed in a "context-based" modelling system. A new (hypothetical) context could be created, G assumed to be true, implications of this assumption determined, and any occurring violations discovered. If no violations were discovered, the hypothetical context could be popped and the implications about the assumption of G true would be discarded.

The dependency scheme never throws away the work it has done in computing the implications of a given truth value for an assertion. When G actually becomes true as a result of some action proposed by the planning process, much of the effort expended by inference to determine the implications of the assumption about G being true can still be used.

In addition to its efficiency, I would argue that this is a more general violation detection scheme than has been proposed before (see References 2, 3 and 5, for instance). It is my feeling that utilizing a general inference process for violation detection allows for application of the problem solver to more complex domains than would be allowed by a detection process which relies solely on syntactic pattern matching. Furthermore, the dependency structure helps focus the search for a violation.

Descriptor binding

It is often necessary during problem solving to defer decisions about the bindings of certain variables. These are variables for which bindings are not explicitly specified when a strategy pattern (Figure 19) is instantiated to satisfy a goal statement (Figure 20). Thus, the "CLEARTOP" strategy has a free variable which should be bound to the

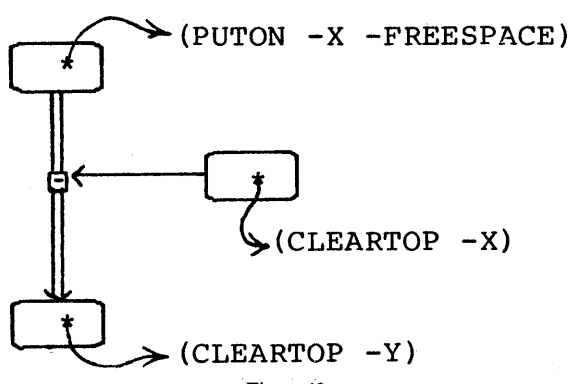


Figure 19

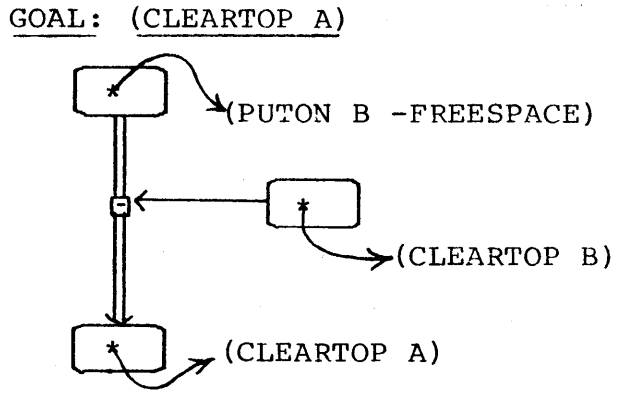


Figure 20

location at which the block on top of the block to be cleared should be placed. One advantage of being able to defer the choice of a binding for this variable is that it may not be obvious at the time the strategy pattern in Figure 19 is instantiated what the most appropriate choice for a binding would be. By deferring the decision, the problem solver can wait until enough information about a suitable binding has surfaced before making that decision.

Consider the problem from Waldinger[3] depicted in Figure 21. For the purpose of synthesizing a reasonably efficient plan, the problem solving system should have the ability to detect that in clearing A, the most suitable location to place B is ON C, thus solving the second subgoal in the process. Problems of this complexity have been handled by defining variables which can be manipulated by the problem solver as objects, and binding those variables to actual objects at the time at which a suitable determination of the correct binding can be made.[5,7]

The solution which I am proposing follows the same line of reasoning by binding variables which can be treated as objects. These variables are called *descriptors* because they can describe semantic restrictions on the eventual referrent. For example, consider the descriptor

```
(*DESCRIPTOR -FREESPACE
    [OR ⟨AND (CLASS -FREESPACE BLOCK)
             (CLEARTOP -FREESPACE)⟩
        (REGION-OF -FREESPACE TABLE)]
    [TABLE-REGION-1] ).
```

This descriptor describes the object on which to place a block which has been picked up for the purpose of clearing another. It is a variable with semantic restrictions on its eventual binding specifying that FREESPACE is either a block whose top is clear or it is some region of table. If no

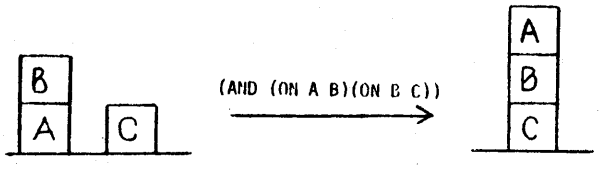


Figure 21

binding can be discovered during plan synthesis, the table is declared as the default.

The relationship of descriptor binding to the dependency network is as follows. All assertions which refer to a descriptor are included in the dependency net. In the example above, the assertion (ON B -FREESPACE):T would be included in the net as a result of clearing A when achieving (ON A B). If the problem solver makes a request for the truth of a goal (e.g., (ON B C)) which matches an assertion in the dependency net containing reference to a descriptor (e.g., (ON B -FREESPACE)), then a candidate for a descriptor binding has been found. The potential binding object (C) must still pass a few tests. If the object which matched successfully against the descriptor also satisfies the semantic restrictions specified in the descriptor, and it satisfies all other assertions in the net which reference the descriptor, then the object can be bound to the descriptor. This binding technique is similar to Sacerdoti's "Use Existing Object" critic.[5] A scheme is currently being considered to allow binding to occur even though not all the restrictions are met, and to allow the satisfying of those restrictions to be solved as subgoals by the problem solver.

Descriptor binding amounts to an assumption made by the problem solver about an appropriate object to participate in the plan. The choice must be carefully determined, and will depend on facts known to the system. As such, the decision can be knitted into the dependency net as a new type of dependency. The advantage of making descriptor binding explicit within the dependency framework is that it allows the descriptor binding mechanism the same flexibility as other inference processes in the system. The choice of descriptor binding is considered best at the time the choice is made. Subsequent problem solving might prove that choice inferior to another. Since the binding is just a type of dependency, these decisions can be "unravelled" as easily as action sequence decisions to be described in the next section. Furthermore, the network treats descriptors as objects and can perform inference based on the features specified in the descriptor as though it were a real object.

Modifying the action sequence

A common operation in problem reduction problem solvers is to modify the partial action sequence as a plan is being synthesized. This modification is most often utilized by "sequential" type systems (those which deal with the subgoals sequentially) and is applied to relieve certain types of subgoal interactions. The particular types of interaction which can be relieved in this way are those which result from an incorrectly ordered action sequence (a possible side-effect of the assumption of subgoal independence).

When subgoals are attacked independently, it is difficult to anticipate the proper order in which they should be solved. Often, in fact, an inherent interdependence among subgoals demands that the solutions that are independently derived for each be somehow melded together. A classic example of subgoal interdependence is depicted in Figure 22. Regardless of the order in which the subgoals (ON A B)

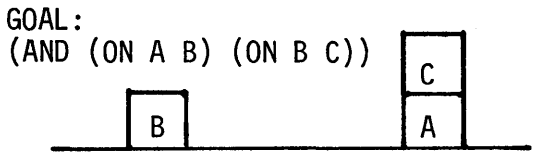


Figure 22

and (ON B C) are solved, a reasonable plan cannot be synthesized by considering the two subgoals independently. A reasonable approach is to allow the subgoals to proceed sequentially and if a subgoal is violated by a subsequent subgoal to modify the action sequence to relieve the violation.*

Operations such as inserting a new action in the middle of an existing action sequence or removing an action from the middle of an action sequence must be acceptable to the modelling mechanism. This is a task which is difficult for chronological "context-based" models to deal with. The purpose of the modelling mechanism is to reflect the anticipated state of the world as it would be when modified by the sequence of actions being proposed by the problem solver. Let us examine an action sequence modification which involves removing an action, Ai, from an action sequence $\{A1, \ldots, An\}$, where $n>i$. The action-consequent links for the actions $Ai, \ldots, An$ can be removed and the inference mechanism can update the network to reflect these changes. The effects of actions $Ai+1$ through An can then again be accommodated efficiently, making use of the already existing network structure created when they were originally introduced.

Frequently the effects of actions are very local in the sense that other actions have little effect on the assertions effected by a particular action.** The power of the DB model for the operation of action sequence modification is derived from the fact that, often, many of the effects of the actions subsequent to Ai are little changed by the removal of Ai from the sequence or the insertion of a new action into the sequence. Since the effects of actions are explicitly represented in the dependency net, much of the effort necessarily expended by a chronological model in recovering to a state of consistency after removal (or addition) of an action from the action sequence can be avoided.

Let us examine some of the problems of using a chronological "context-based" model for these operations. Figure 23 shows a sequence of context frames in a context-layered database (a la CONNIVER[14]) where each frame represents the assertions required to accommodate subsequent actions in the proposed action sequence $\{A1, \ldots, An\}$. An efficient operation for the context-based models is to revert to a prior state, say the state just before inserting Ai on the action sequence (Figure 24). This operation would be necessary if one desired to remove Ai from the action sequence (and its effects from the model). However, if one now tries to include the subsequent actions in the model (excluding Ai), all the work already performed in accommodating $Ai+1$ through An must be reproduced (Figure 25). This is because context-layered databases allow a tree-like structure of context frames (as in Figure 25) while what is ideally required is a graph structure (Figure 26).*

Modelling mechanisms other than "context-based" exist which handle the action sequence modification problem. Most notable are the "plan-structure" schemes of Sacerdoti[5] and Waldinger.[3] Both these schemes make use of the existing plan for computing the model. Both offer no more flexibility than dependency nets for recovering to prior states but they suffer, as do the context models, from being "forgetful" about inferences which have been made since the point at which the action sequence is being modified. In Waldinger's model, this forgetting is deliberate because he does not want the problem solver to be committed to belief in an assertion derived on the basis of the current structure of the plan. The lack of commitment to derived assertions is required since the plan structure on which the derivation was based may be altered by a modification to the action sequence. Dependency nets allow the possible alteration of belief in an assertion due to modification of the plan to be explicitly detectable. Thus, beliefs whose derivations are based on the current structure of the plan can be incorporated into the DB model (whereas in a plan-structure model they cannot) because the dependency net provides a means to detect which beliefs are affected by a modification to the plan.

FUTURE WORK AND IMPLEMENTATION STATUS

The ideas presented here are currently under implementation. Among the immediate problems is that of restricting the bindings of variables for instantiations of selected support rules. It is often possible for the consequent of a rule to match a newly entered assertion, while its antecedent matches no assertion in the net. This can leave unbound variables in assertions for which truth values are believed. For the rule in Figure 27, the consequent (CLEAR B):F matches (CLEAR -Y) and so (ON—XB):* is asserted. I plan to use the descriptor mechanism to specify the range of values the universally quantified variable—X might be bound to. Further, I intend to investigate the possibility of

* There are several schemes for determining an appropriate action sequence modification to relieve subgoal protection violations. See References 1-4.

** See Stallman and Sussman[9] for an analogous discussion of the influence of particular electronic components on the overall behavior of a circuit.

* It should be emphasized that context-based chronological models are more efficient for certain operations than the DB model. These operations are (1) the recovery to prior states (discussed above) and (2) the ability to transfer from one line of reasoning to another. For example, if it is later discovered that it would have been better to leave Ai in the action sequence, a context-based model provides for efficiently returning to that line of reasoning (i.e., transferring from point X in Figure 25 to point Y). These operations cannot be performed as efficiently by dependency net models. A possible compromise (which has not been pursued) would be to implement the dependency net in a context layered database for the specific purpose of making these desired operations efficient within the dependency net domain. It would be, however, a space-wise inefficient implementation, compounding the space-wise inefficiencies of both dependency nets and context layered databases (e.g., CONNIVER).

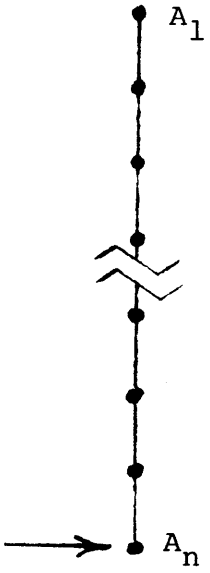


Figure 23

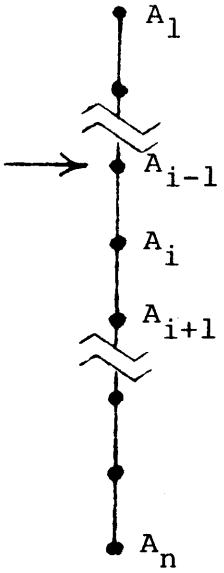


Figure 24

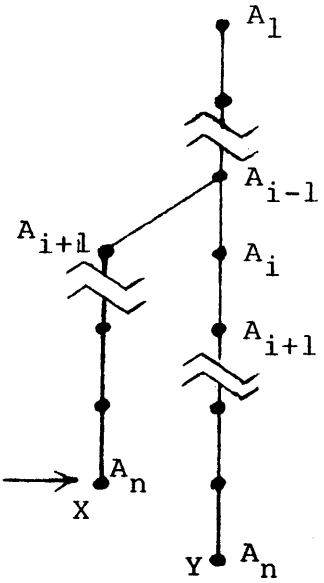


Figure 25

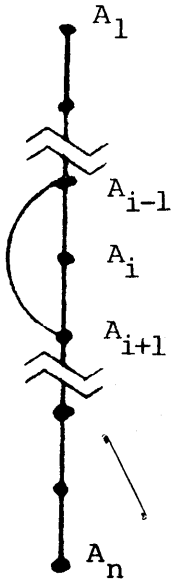


Figure 26

invoking the non-monotonicity assumption of Doyle[11] in order to limit the requirements for redeductions of truth values for assertions which have lost all support.

The implementation strategy I have adopted is to associate with each arc in the net a set of associatively triggered computation units (i.e. demons),[15] whose purposes are to propagate information across the arcs when appropriate. Making the dependency net links be active components is an idea borrowed from the CSA mechanisms simulator.[18] In my case, this technique has the advantage of allowing the implementation of network capabilities to proceed in a very modular fashion. An alternative would be to provide a program which interprets a declarative representation of the network, but would require the integration of all the network capabilities into a single program.

CONCLUSIONS

This discussion has centered on dependency nets as a unified formalism for modelling in general problem solving systems. I have suggested how such a modelling system behaves and how some enhancements to the basic ideas allow them to operate efficiently. Dependency nets serve to limit the amount of deduction required while performing inference for determining the important consequences of acquiring or modifying beliefs.

Various competing solutions have been presented for some of the problem solving difficulties discussed. Most were presented within the context of complete systems with well defined capabilities. I feel that enough has been learned about the problems of general problem solving that some investigations can be initiated to research specific techniques independent of any particular complete problem solving system. This endeavor demands some formalism in which these techniques can be described. I intend to pursue the idea that for many problem solving problems, dependency nets are an important formalism.

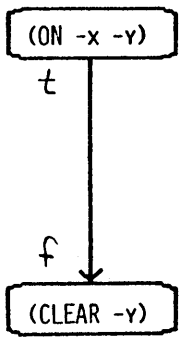


Figure 27

ACKNOWLEDGMENTS

I wish to thank members of the Commonsense Algorithm Research Group—Chuck Rieger, Milt Grinberg, Steve Small, Rich Wood, and George Fekete—for many discussions and help in preparation of this document. I also wish to thank the National Aeronautics and Space Administration for their support of this research under Contract NSG-7253.

REFERENCES

1. Rieger, Chuck and Phil London, "Subgoal Protection and Unravelling During Plan Synthesis," *Proc. IJCAI5,* Cambridge, Mass., August 1977.
2. Tate, Austin, "INTERPLAN: A Plan Generation System which can Deal with Interactions between Goals," MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh, December 1974.

3. Waldinger, Richard, "Achieving Several Goals Simultaneously," SRI AI TN #107, July, 1975.
4. Warren, David, "WARPLAN: A System for Generating Plans," Memo No. 76, Department of Computational Logic, University of Edinburgh, June 1974.
5. Sacerdoti, Earl D., "A Structure for Plans and Behavior," SRI AI TN #109, August 1975.
6. Dawson, Clive and L. Siklossy, "The Role of Preprocessing in Problem Solving Systems," *Proc. IJCAI5,* Cambridge, Mass., August 1977.
7. Sussman, Gerald J., "A Computer Model of Skill Acquisition," MIT AI-TR 297, Aug. 1973.
8. Fikes, Richard E., "Deductive Retrieval Mechanism for State Description Models," *Proc. IJCAI4,* Tblisi, USSR, September 1975.
9. Stallman, Richard M. and Gerald Jay Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer Aided Circuit Analysis," MIT AI Memo 380, September 1976.
10. Duda, Richard O., Peter Hart and Nils Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," SRI AI TN #124, January 1976.
11. Doyle, Jon, "Truth Maintenance Systems for Problem Solving," MIT AI TR-419, 1977.
12. Rieger, Chuck, "The Commonsense Algorithm as a Basis for Computer Models of Human Memory, Inference, Belief, and Contextual Language Understanding," *Proc. of Workshop on Theoretical Issues of Natural Language Processing,* Cambridge, Mass. 1975.
13. Chang, Chin-Liang and Richard C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, 1973.
14. McDermott, Drew V. and G. J. Sussman, The CONNIVER Reference Manual, MIT AI Memo 259a, Jan. 1974.
15. Rieger, Chuck, "Spontaneous Computation in Cognitive Models," *Cognitive Science,* Vol. 1, No. 3, Fall 1977.
16. McCarthy, J. and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence,* Vol. 4, B. Meltzer and D. Michie, eds., American Elsevier, New York, 1969.
17. Fahlman, Scott, "A Planning System for Robot Construction Tasks," MIT AI-TR 283, May, 1973.
18. Rieger, Chuck and M. R. Grinberg, "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms," *Proc. IJCAI5,* Cambridge, Mass., August 1977.

Error recovery in robots through failure reason analysis*

by S. SRINIVAS*
California Institute of Technology
Pasadena, California

INTRODUCTION

In performing a task, a robot may encounter a variety of unforeseen circumstances which prevent successful completion of the task. Even simple actions, such as reaching out to grasp a rock, cannot be guaranteed to succeed without dynamic sensory feedback such as visual monitoring. For example, the hand may bump into the rock because of manipulation errors or because of erroneous information about the location of the rock. Failures such as these occur because sensors and effectors are inaccurate, and because the world in which the robot operates cannot be characterized exactly.

Our goal is to design a robot that can recover gracefully and intelligently from failures that occur during the execution of tasks in a static world with no other agents of change. The technique for recovery planning discussed in this paper is based on knowledge about failures—why they occur, what can be done about them, etc. This knowledge is used to determine the cause of a failure and to devise a recovery plan. Similarities and differences between this and other approaches are discussed in a subsequent section.

A computer program called MEND has been implemented for exploring the above concepts. Though MEND only automates recovery from failures in simple manipulation tasks that are performed by the JPL robot,[8,9,17–19] the techniques used in MEND are more widely applicable.

A DISCUSSION OF THE APPROACH

Execution of a task can be viewed as a sequence of transformation that changes the world from an initial state, S_0, into a final state, S_n, which represents the goal of the robot.

Figure 1 depicts the intermediate states of the transformations from S_0 to S_n that result from the execution of a plan of actions, $A_0 \ldots A_{n-1}$. Assume that a failure is detected on execution of action A_i. This means that the robot recognizes that action A_i has not produced the expected state S_{i+1}, but rather has resulted in some failure state S_f.

The problem of error recovery is that of going from the failure state S_f to the goal state S_n. One approach is to treat S_f as though it were some arbitrary state and respond to the failure by replanning to achieve the given goal. In such an approach, there is no essential difference between error recovery [the transformation from S_f to S_n] and planning [for instance, the transformations from S_0 to S_n]. Parts of the old plan can sometimes be reused, as for instance, by replanning from S_f to S_j so that the original plan can be used to go from S_j to S_n.

This paper suggests that by finding an explanation of the failure, i.e., understanding why action A_i resulted in the state S_f, a robot can determine where the problem lies and what can be done about it. With this viewpoint, error recovery is seen as something quite different from traditional planning in that we do not merely ask, "What can be done to go up from S_f to S_n?," but begin by asking, "Why did action A_i result in the state S_f?" in order to recover from the failure.

OVERVIEW OF MEND

Let us consider an example to illustrate the problems involved in recovery from failures, and some of the features of the solution incorporated in MEND. The robot hardware that is controlled by MEND consists of two TV cameras, a laser rangefinder, a manipulator, and a vehicle (Figure 2).

Suppose that the robot is given the simple task of picking up a rock within immediate reach. Performing this task involves locating the rock using the TV cameras, opening the hand, positioning the hand around the rock, and grasping the rock (Figure 3). Assume that the hand bumps into the rock when attempting to position the hand around the rock. In this example, the manipulator system triggers the failure, bringing MEND into action. In other cases, MEND has to make explicit checks of certain conditions to detect failures. MEND's execution monitoring is pragmatic in the sense that only those conditions that are easy to check on the basis of the information available directly from feedback, sensors, etc., are tested. A consequence of this lack of comprehensiveness is that failures can propagate and may be detected only at a subsequent step.

Having detected a failure, MEND is confronted with the

* Now employed at Bell Laboratories, Naperville, Illinois.

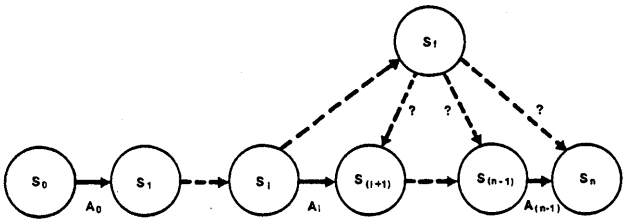


Figure 1

problem of dealing with the unexpected event. A heuristic used in MEND suggests that recovery actions can be found by determining why the failure occurred. The process of finding an explanation for the failure is termed *failure reason analysis*. Knowledge necessary for failure reason analysis is provided through a *failure reason model* associated with each action. The failure reason model represents knowledge about different reasons for failure of an action, how they may be distinguished, and what can be done to recover from a specific kind of failure.

MEND represents possible explanations for a failure in a structure called the failure tree, an example of which is shown in Figure 4. The tree consists of a linked set of failure nodes and action nodes. There are four basic types of failure reasons that are dealt with in MEND; namely *operational errors, information errors, precondition errors,* and *constraint errors*. Each failure node is associated with a failure type and a specific reason for failure. Action nodes are linked to an action in the plan being executed. An example of an explanation represented in Figure 4 is that the goal WITHIN-GRASP (node G) was never achieved by action MOVE-HAND-TO-GRASP, because of an operational SERVO-ERROR (node F1).

Analysis of failures can now be thought of as the process of limiting the set of all possible explanations to the specific one that applies in a particular situation. MEND does this by limiting the set of failure reasons to be considered at each action node in the failure tree. Preconditions and constraints that have been verified on the basis of feedback information and sensor outputs, can be eliminated from consideration as

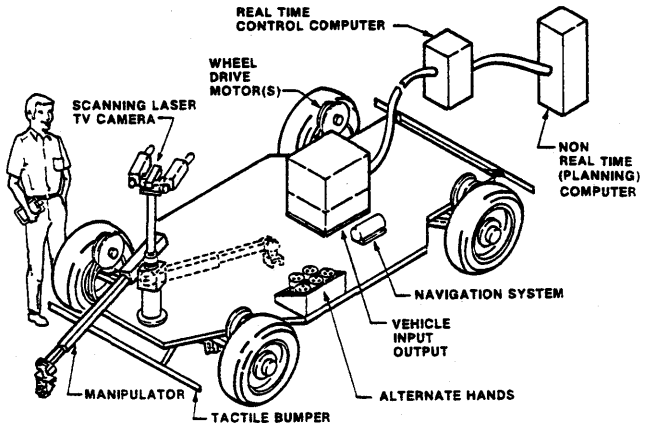


Figure 2

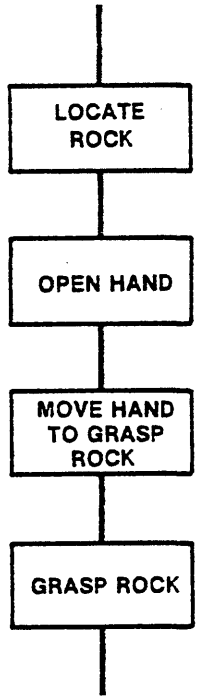


Figure 3

possible reasons for failure. More importantly, the distinctive features of the failure are used to limit the set of failure reasons. Figure 5 shows, for instance, a simple test and the implications of its results in eliminating some failure reasons from consideration.

Once an explanation has been found, recommendations of corrective steps can lead MEND to successfully plan recovery from failures. For instance, if the failure is attributed to an information error (Figure 4) recovery can be achieved by

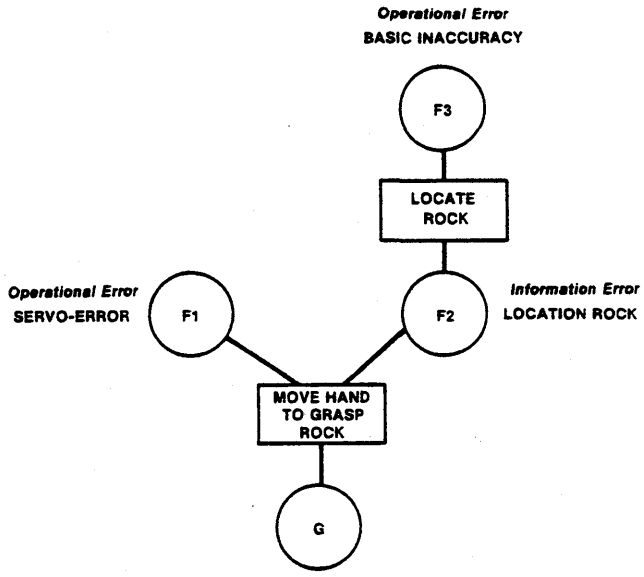


Figure 4

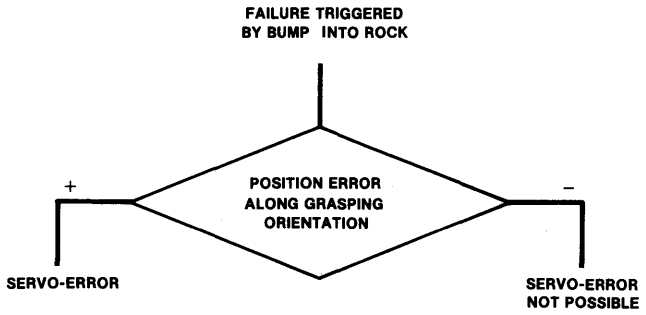


Figure 5

positioning the hand near the rock, determining the location of the rock relative to the hand, and then repositioning the hand around the object (Figure 6).

Other scenarios illustrating MEND's capabilities are described in a thesis[14] that is a more complete documentation of the research study discussed here. The following sections describe the conceptual basis behind MEND and its design.

A CLASSIFICATION OF FAILURE REASONS

MEND's analysis of failures is based on a classification of failure reasons into four categories. These are operational errors, information errors, precondition errors, and constraint errors. Each is discussed in turn.

Actions can fail to achieve their intended result because of certain inherent problems in executing the action. These errors are peculiar to the specific operation being performed and are, therefore, referred to as operational errors. For instance, the manipulator may deviate from the planned trajectory in moving from one location to another. This servoing error will not often have any serious consequences, but in some cases it can cause the manipulator to bump into the object to be grasped.

An inaccuracy in the location of an object is an example of an information error. Though information errors can usually be traced back to operational errors in the process being used to provide the information, they are to be distinguished from such operational errors. As in our previous example of operational errors, information errors will not always cause failures and the robot may successfully complete its task despite such errors.

Before executing an action, MEND will check preconditions and constraints. If any of these are found to be false, it will attempt to make them true before continuing. If all the preconditions of actions could be checked and verified, there would be no reason for failures to occur because of precondition errors.

However, it would be unrealistic to expect a robot to check preconditions exhaustively, since some of them are inherently difficult to check. Therefore, the execution monitoring in MEND does not require that all preconditions be absolutely verified before executing an action. In some cases, executing subsequent steps may be the simplest way of finding out that they were or were not satisfied. For instance, checking to see that an object is truly within grasp before actually grasping the object is unnecessary and expensive, since the very action of grasping will immediately indicate whether this is true or not. Even when previous actions were executed with the intention of making the preconditions true, the robot cannot depend on them being true since these actions themselves may fail. In any case, the consequence of not verifying preconditions is that *after* execution of an action, failure can be attributed to the falsity of preconditions. In this report, the term precondition errors refers to the above interpretation, to be distinguished from precondition failures detected before execution of an action.

Finally, an action can fail because there are some constraints that must be satisfied for successful execution. An example of this is that the object should be movable before the robot can transport it from one place to another. In a sense, these are merely special cases of preconditions that the robot has no way of making true.

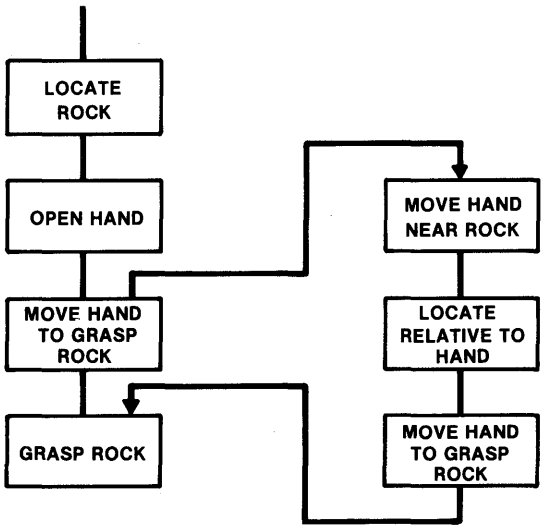


Figure 6

THE FAILURE TREE

The structure of a failure tree, as illustrated in Figure 7, is described next. There are two types of nodes in a failure tree, *failure nodes* and *action nodes*. Failure nodes are represented by circles and action nodes by rectangles. Action nodes are linked to a node in the execution trace through a plan link that is not shown in the preceding figure. Each failure node identifies the failure and its type, and may point to one or more action nodes through one of several kinds of links. The NAB (Never Achieved By) and IPB (Incorrectly Provided By) are two examples of such links. Action nodes point to failure nodes through a PRFF, the "Possible Reason For Failure" link.

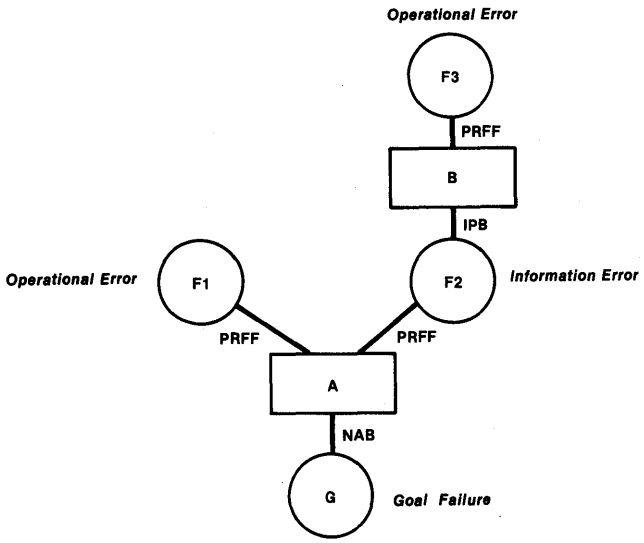


Figure 7

The example in Figure 7 can now be interpreted in the following manner: Goal G was never achieved by action A because of one (or more) of two possible reasons, F1 and F2. F1 indicates that the failure could have been the result of operational error, while F2 represents data incorrectly provided by action B.

The types of failure associated with failure nodes are the ones discussed earlier (operational, information, precondition, and constraint errors) with one addition. There is a special failure type which occurs only at the root of the tree and this represents a goal failure.

Operational and constraint failure nodes do not point to other nodes since they are local to the failure action. Precondition nodes may be linked to action nodes that were intended to achieve the precondition. When such links are missing, it means that no action was executed to achieve this condition. It, therefore, represents an *initial condition error*. Similarly, information failure nodes are linked to actions that provide the needed information, and missing links indicate *initial information errors*.

BUILDING A FAILURE TREE

The idea behind failure reason analysis is to find an explanation of the failure. By *explanation* we mean a chain of reasons represented by a path from the root node of the failure tree to one of its leaf nodes (G - A - F2 - B - F3 in Figure 7). If we consider all possible reasons for failure at each action node, then the failure tree represents all possible explanations for the failure. Clearly, only one or a few will be relevant in any particular situation.

We next investigate what can be done to constrain the set of explanations. An execution trace in which unverified preconditions and constraints are noted provides the information necessary for determining whether failure can be attributed to the falsity of these conditions. Note that a precondition is not considered to be verified merely because a previous action was executed to make this condition true. Instead, it is considered to be true when independent tests verify the truth of the precondition. To consider a trivial example, OPEN-HAND is verified by checking that the joint feedback information indicates that the fingers are fully separated. Typically, other robot systems verify preconditions by looking for explicit assertions in the data base or by deducing the truth of these conditions from other assertions. The result of this is that no distinction can be made between those preconditions that can be verified directly from feedback information and those that are only surmised to be true because of the intended effects of previous actions.

A second important way in which the failure nodes branching from an action can be cut down is by looking at the manifestation of the failure. By identifying distinctive features of the failure situation, some of the reasons for failure can be shown to be impossible or at least unlikely.

Third, the history of actions can show that certain explanations are impossible. If, for instance, no reasons for the failure of action C in Figure 8 can be found, we can conclude that B could not have failed because of F2. If there are no other reasons for the failure of B besides F2, we can conclude further that A could not have failed because of F1. Such reasoning can significantly cut down the set of possible explanations, thereby pointing out the real explanation of the failure.

Failure reason analysis in MEND, i.e., the process by which MEND determines the plausible explanations of the failure by building a failure tree, is outlined below:

1. Find the set of possible failure reasons.
2. Eliminate verified preconditions and constraint failures from this set by looking at the execution trace.

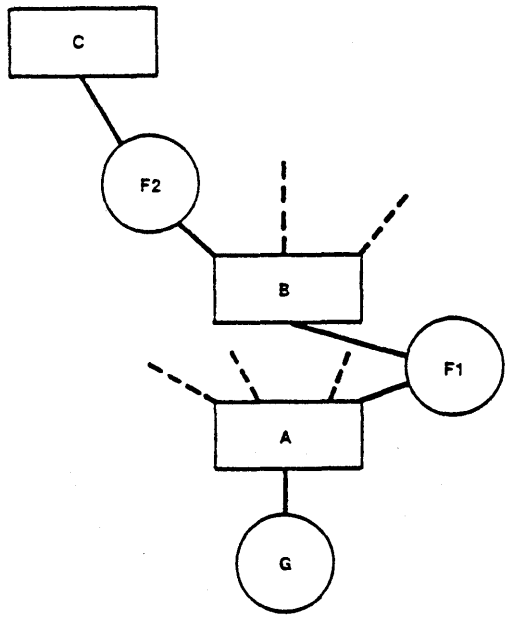


Figure 8

3. Look for distinctive features of the manifestation to further constrain the set of failure reasons.
4. For each of the remaining failure reasons, create failure nodes and link the action node to these newly created failure nodes.
5. For precondition failure nodes, find the previous action responsible for making the precondition true from the execution trace. For information failure nodes, find the previous action that provides the necessary information. Create an action node for each of the previous actions so determined.
6. Repeat the above process for the newly generated action nodes.
7. Clean up the failure tree by eliminating impossible explanations (i.e., apply the process described in connection with Figure 8).

THE FAILURE REASON MODEL

The failure reason model is a model of actions which provides the knowledge necessary to perform failure reason analysis by means of the algorithm outlined in the previous section. First, the model should identify the set of possible failures. Operational failures are represented explicitly through a set of associations as shown below:

(OPERATIONAL-ERROR MOVE-HAND-TO-GRASP(ROCK) SERVO-ERROR)
(OPERATIONAL-ERROR MOVE-HAND-TO-GRASP(ROCK) COLLISION)
(OPERATIONAL-ERROR LOCATE(ROCK) INACCURACY)

Knowledge about other types of failures is implicit in the model of actions used in MEND, which specifies preconditions, constraints and needed information. For instance, any unsatisfied precondition is recognized as a possible reason for failure by MEND. Figure 9 shows the correspondence between the action model and the failure tree.

The knowledge necessary to eliminate some possibilities on the basis of their manifestation is highly domain dependent. In MEND this knowledge is incorporated in a routine called the FRA-ROUTINE which is associated with each action. The FRA-ROUTINEs test for the presence or absence of distinctive features of the failure in the current state of the world. These tests indicate that certain failure reasons are impossible, that others are likely, and so on.

Figure 10 represents the FRA-ROUTINE associated with the action MOVE-HAND-TO-GRASP. At each node in the tree, a condition is tested and the appropriate branch taken. The leaf nodes of the tree specify a list of failure reasons that are considered possible in a specific context.

We note several important points about these routines. As with precondition testing, the conditions being tested are easily computed using the data in the world model and the feedback information. For instance, the condition NEAR-DESTINATION is checked by verifying that the final position of the hand is within a prespecified limit of the planned destination.

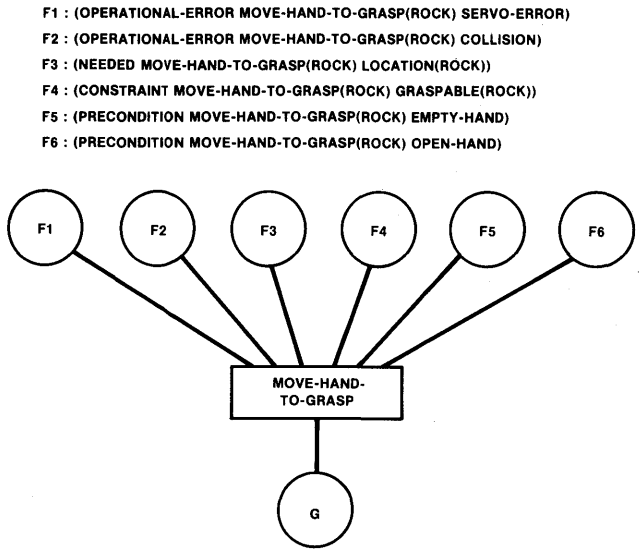


Figure 9

The routine illustrated in Figure 10 assumes that the preconditions EMPTY-HAND and OPEN-HAND have been made true, and does not consider how a failure resulting from the falsity of these conditions will manifest itself. This simplification is possible because MEND's execution monitoring will detect these precondition failures and will not allow execution of this action to continue unless the preconditions have been satisfied. Note that the FRA-Routines represent a procedural incorporation of knowledge about the manifestations of failure.

There is a problem in the use of such routines to limit the set of failure reasons in Step 3 of the algorithm presented in the previous section. This routine requires information about the state of the world in making its decisions, which means that previous states of the world need to be represented. To keep this information around would be quite unrealistic in any large system, even using a stack-like mechanism for representing only the incremental changes.

There are two ways of tackling this problem. Execution monitoring can be extended by running the FRA-ROU-

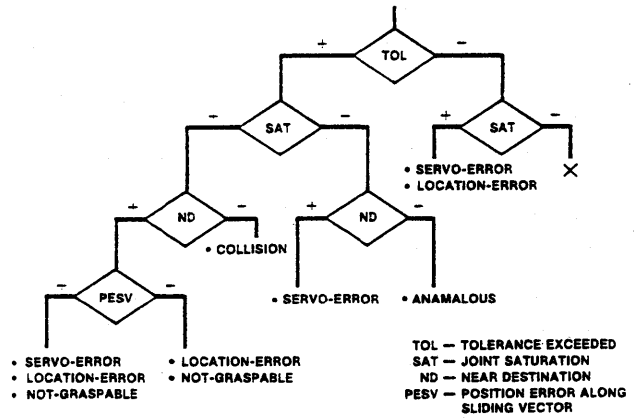


Figure 10

TINEs regardless of whether or not a failure occurred and marking the failure possibilities in the execution trace. Later, if failure reason analysis is necessitated in a subsequent action, this possibility list presents a summary useful for analysis.

A different strategy is to ignore the possibility of failure when there is no explicit triggering of the failure, and to later determine the failures that escaped detection. This is achieved in MEND by the representation of facts such as the one shown,

```
(PTRIG (NEEDED MOVE-HAND-TO-GRASP(ROCK)
  LOCATION(ROCK)) NTRIG)
where  PTRIG=POSSIBLE-TRIGGER
and    NTRIG=NULL-TRIGGER,
```

which says that a location error in the position of the rock could go untriggered when executing MOVE-HAND-TO-GRASP. (The reason for this is that if the location error is sufficiently large, the hand will completely miss the object.) With these facts in hand, MEND can immediately determine the set of failure reasons to be considered in Step 3 of the previously outlined algorithm, without having to run the FRA-ROUTINE for previously executed actions.

An assumption implicit in the latter approach is that any triggered failure has already been handled. The first approach is a more general technique but will perform unnecessary computations in situations where plans are successfully executed.

There is one other aspect of the failure reason model. It provides information about how to correct a particular kind of failure.

Knowledge necessary to correct operational errors is local to an action, and is represented in MEND by facts, such as,

```
(TO-CORRECT (OP-ERROR MHTG(ROCK) SERVO-
  ERROR) MHTG-REC-PLAN1(ROCK))
where  OP-ERROR=OPERATIONAL ERROR
       MHTG=MOVE-HAND-TO-GRASP
and    MHTG-REC-PLAN1=(ROCK)
                      BEGIN
                      MOVE-HAND-TO-
                      GRASP(ROCK)
                      END,
```

which says that an operational error of servoing in MOVE-HAND-TO-GRASP can be corrected by executing the plan MHTG-REC-PLAN1. This plan will attempt recovery by simply repositioning the hand, and this strategy is appropriate for servoing errors. In general, TO-CORRECT steps are provided for each operational error.

Constraint errors on the other hand, have no such "imperative knowledge"[6] about what to do, since the robot does not have the capability of correcting such failures. If failure is attributed to a constraint error, the plan is aborted and MEND seeks human aid.

Precondition and information errors are not directly associated with corrective steps and are treated somewhat differently than operational errors. For precondition errors, the obvious thing is to reachieve the failed precondition before trying the action again. But before this can be done, some of the effects of the failed action may have to be undone. After the precondition has been achieved, other steps may need be redone to get execution back on the right track. MEND implicitly incorporates this knowledge and recognizes that there are three parts to recovery from precondition errors: *undo certain actions, reachieve precondition, and redo the undone steps.* Knowledge about what needs to be undone and what needs to be redone is part of the failure reason model. For example,

```
(UNDO-STEPS (PRECONDITION GRASP (ROCK)
  WITHIN-GRASP (ROCK)) OPEN)
(REDO-STEPS (PRECONDITION GRASP (ROCK)
  WITHIN-GRASP (ROCK)) GRASP(ROCK)),
```

says that if failure in GRASP is attributed to the precondition error of the object not being WITHIN-GRASP, the OPEN undoes the effect of GRASP, and that GRASPing the object needs to be redone after WITHIN-GRASP has been reachieved.

The situation is entirely similar for information errors, and recovery is achieved by undoing certain actions, getting the necessary information, and redoing the undone steps. Facts such as these are used in patching together a recovery plan in a rather simple and direct manner, after that the explanation of the failure has been determined. When multiple explanations remain even after analysis, MEND applies an ad hoc technique using severity codes to identify the more serious failure reason, in the hope that the recovery plan will work even if the explanation is wrong.

PERSPECTIVE AND RELATED WORK

Before summarizing the work described in this paper it would be worthwhile to consider related work and place this work in perspective.

Several studies that have dealt with the problem of planning have adopted the paradigm of devising a simple plan for achieving the given goal, identifying and characterizing the problems in such a plan, and finally correcting the problems in the original plan. It is fairly clear that techniques used in systems adopting such a paradigm have applicability to the problem of error recovery.

Sussman,[15] in his investigation of skill acquisition, characterizes failures in terms of domain-independent goal interactions such as precondition conflict between conjunctive subgoals, and the system corrects these by strategies such as reordering actions in the plan, introducing preparatory steps, etc. Sacerdoti[13] and Tate[16] use similar characterizations of subgoal interactions in their planning systems, but the process by which the characterization is achieved has been improved and made more systematic.

In MYCROFT,[6] violations between the actual performance and the intended line drawing described by a model are determined, and then used in correcting the plan. Similarly, Fahlman[1] uses domain-dependent stability analysis to identify instabilities in blocks world structures, and corrects

these by the use of subassemblies, counter weights and scaffolding.

We note the following points that identify the similarities and differences between MEND and the systems mentioned above.

A. In a general sense all of these systems have the capability of dealing with execution time failures. However, Sacerdoti's system NOAH is the only one of these that deals specifically with error recovery. In aiding an inexperienced human to correct an error, NOAH performs a hierarchical search for more detailed descriptions of the incorrectly performed task to identify the failure.

   In contrast to the above systems, MEND more directly models why actions fail *during execution,* and characterizes failure in terms of operational errors, information errors, precondition errors, and constraint errors. This gives MEND the capability of handling real world problems such as inaccuracies in effectors and sensors.

B. MEND uses a data structure called a failure tree to represent explanations of the failure. From a structural point of view the failure tree is domain independent, and reflects the interrelationships between different steps in a plan. Further, it allows domain-dependent details to be attached to nodes in the tree. This structure should therefore have applicability in characterizing failures in many domains.

C. MEND's domain-independent scheme for failure reason analysis identifies the kind of reasoning necessary for analysis of execution time failures, and shows how this type of reasoning can be incorporated in a computer controlled system that performs tasks in the real world.

D. It identifies the kind of knowledge about the domain necessary for performing failure reason analysis, and suggests how such knowledge can be incorporated in a failure reason model associated with each action.

E. It further shows how knowledge about steps needed to correct failures can be used in conjunction with the results of failure reason analysis to effect recovery by means of very simple planning strategies.

LIMITATION AND EXTENSIONS

Failure reason analysis as it is implemented in MEND can certainly be improved and made more general by including subgoal interaction problems. However, MEND assumes that it has been given logically correct plans and does not need to consider problems arising either from incomplete knowledge about the world or from ignoring interactions in the first order solution to a problem.

Another dimension for improvement in failure reason analysis is in handling mechanism failures. An extension in this direction is possible by using a representation of mechanisms such as the one suggested by Rieger.[12] We could also imagine extending failure reason analysis to handle dynamic worlds, but such extensions are likely to be difficult since we know little about how to deal with activities of other agents who may or may not cooperate in the execution of tasks.

An important problem which MEND does not directly address is that of new information that may invalidate subsequent steps in the plan. Though MEND will fumble along by correcting the failure as it occurs, it is short-sighted in not anticipating failures. Hayes[7] has dealt with this problem by means of a representation of plans which makes explicit the dependency between subgoals and decisions made in arriving at these subgoals.

CONCLUSIONS

The study described in this report has shown that extending the modeling of actions to include a failure reason model allows a robot to address itself more directly to the problem of error recovery. The failure reason model provides a means of representing the knowledge about why actions fail, knowledge about their manifestation, and finally knowledge about what can be done to correct failures.

A specialized technique for performing common sense reasoning, appropriate for planning recovery from failure, is embodied in failure reason analysis. This analysis uses the failure reason model to build a failure tree to represent possible explanations of the failure. Planning recovery from failures by determining the explanation of the failure has proven to be a direct and simple way of handling error recovery, at least in the simple manipulation tasks that have been considered. The effectiveness of this method in more complex problem domains is yet to be investigated and requires a more sophisticated planning ability than the one incorporated in MEND.

Finally, it is important to note that error recovery through failure reason analysis is but one approach to the problem. Another effective approach based on the effects rather than on the causes of failure has also been investigated as part of this study,[12] and can supplement failure reason analysis in planning recovery from failures.

ACKNOWLEDGMENT

I would like to thank Meir Weinstein for getting me started in this area of research, for pointing out the importance of automated error recovery in robot systems and for providing guidance and direction during the course of this work. My thanks to Shriram Udupa and Scott Roth for serving as the sounding board for many ideas and for providing help in topics ranging from implementation details to design concepts. Finally, my thanks to Len Friedman and Bill Whitney for many discussions that led to a better formulation of the ideas and concepts described in this paper.

REFERENCES

1. Fahlman, S. E., "A Planning System For Robot Construction Tasks," *Artificial Intelligence,* Vol. 5, No. 1, Spring 1974.
2. Feldman, J., et al., "The Use of Vision and Manipulation to Solve the

'Instant Insanity' Puzzle," Second International Joint Conference on Artificial Intelligence, London, September 1971.
3. Feldman, J. and R. F. Sproull, "System Support for the Stanford Hand-Eye System," Second International Joint Conference on Artificial Intelligence, London, September 1971.
4. Fikes, R. E., "Monitored Execution of Robot Plans Produced by Strips," SRI Artificial Intelligence Center, Technical Note 55, April 1971.
5. Finkel, R., et al., "AL, A Programming System for Automation," Stanford AI Memorandum 245, November 1974.
6. Goldstein, I. P., "Understanding Simple Picture Program, MIT Artificial Intelligence Laboratories, Technical Report 294, September 1974.
7. Hayes, P. J., "A Representation of Robot Plans," Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, September 1975.
8. Hooke, A. A., B. T. Larman, W. M. Whitney, "The Impact of Robots on Planetary Mission Operations," International Telemetering Conference, Los Angeles, October 1974.
9. Lewis, R. A., "Practical Manipulator Software," *Proceedings of the IEEE,* Symposium on Automatic Computation and Control, Milwaukee, April 1976.
10. Nilsson, N. J., "A Hierarchical Robot Planning and Execution System," SRI Artificial Intelligence Center, Final Report, October 1971.
11. Raphael, B., et al., "Research and Applications—Artificial Intelligence," SRI Artificial Intelligence Center, Final Report, October 1971.
12. Rieger, C., "One System for Two Tasks: A Commonsense Algorithm that Solves Problems and Comprehends Language," Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, September 1975.
13. Sacerdoti, E. D., "A Structure for Plans and Behavior," SRI Artificial Intelligence Center, Technical Note 109, August 1975.
14. Srinivas, S., "Error Recovery in Robot Systems," California Institute of Technology, Information Science, Ph.D. Thesis, December 1976.
15. Sussman, G. J., "A Computational Model of Skill Acquisition," MIT Artificial Intelligence Laboratory, Technical Report 297, August 1973.
16. Tate, A., "Generating Project Networks," Fifth International Joint Conference on Artificial Intelligence, Cambridge, August 1977.
17. Weinstein, M., "A Framework for Robotic Design," California Institute of Technology, Information Science, Technical Report 14, June 1975.
18. Weinstein, M., "Structured Robotics," California Institute of Technology, Information Science, Technical Report 15, June 1975.
19. Whitney, W. M., "Human Versus Autonomous Control of Planetary Roving Vehicles," IEEE Symposium on System, Man and Cybernetics, Dallas, October 1974.

Area Director:
W. M. vanCleemput
Stanford University
Stanford, California

Design automation

Over the last decade design automation has emerged as an irreplaceable tool for engineering design. The design of today's complex systems such as aircraft, ships, spacecraft, chemical plants, automobiles and computer systems would be unthinkable without extensive computer aids for analysis, simulation and synthesis.

Whereas in the past design automation tools consisted of a collection of loosely connected programs, the current trend is towards an integrated design automation system centered around a common design database.

The last decade also witnessed the emergence of low-cost computer graphics terminals that made it possible to use man-machine interaction in a cost-effective way.

Three sessions explore the various aspects of design automation during this conference. The first session deals with automated design methodologies for large hardware/software complexes. Digital systems have become increasingly complex and new design strategies are required in order to achieve satisfactory designs. The second session deals with the layout of very-large integrated circuits. Due to the explosive development of integrated circuit technology, we are now faced with the problem of efficiently laying out extremely large circuits. The papers in this session explore several ways of solving this problem in an effective manner. Finally, the third session discusses mature design automation systems. In this session we try to look back at some existing systems in order to evaluate their usefulness and impact.

IC design—Misery or magic

by K. J. LOOSEMORE
Pompador Ltd.
Hertfordshire, England

Ever since integrated circuit designers began to put thousands of transistors onto a single chip the cost, in terms of human labor, required to lay out the circuit has been extremely high. If we analyze the layout task we find, in general, that two almost distinct phases are involved. First the designer will design the basic building blocks for the circuit and connect them together. The former of these two phases of design requires considerable expertise and experience, making efficient automation extremely difficult. The second phase, however, only requires lower grade labor, since the efficiency of this part of the layout in terms of size and electrical characteristics is usually less critical. The second phase, none the less, requires a considerable proportion of the design time and it is this fact, together with the need for it to be carried out from scratch for each new design and its lower criticality factor that make it a candidate for automation.

The MAGIC suite of programs has been developed at the UNIVERSITY of EDINBURGH both to address the interconnection problem and to attempt to provide some structuring facilities for the more efficient design of integrated circuits.

The system consists of several programs centered around a main automatic/interactive layout generator. It is this main layout generator which is of most interest but it is worthwhile to give a general description of the rest of the suite in order to see how the system fits together.

The designer begins by taking his circuit diagram and splitting it up into a number of functional modules (e.g., a stage of a shift register or part of a decoder). Specifications for these functional modules are then written in the system's input language. At this stage there are only two requirements, first the rectangular outline of each module must be specified and secondly the positions of the interconnect points (called "pins") to the modules, must be defined. It is not necessary at this point to have the logic internal to the module defined. This is required only at the final output stage of the suite.

Bonding pads around the outside of the chip are designed in a similar manner, however, their positions around the chip are vaguely specified (only side allocations and order are required). These definitions, once produced, would normally be kept in a library file (Refer to Figure 1) and made available for future designs.

After specifying the logic modules the details of their interconnection are defined. For each connection net a list of connections is given and, in order to minimize errors at this point, it is also necessary to include a list of all defined connection points which for this particular design are not connected. This feature allows the program to detect accidental non-connections.

In addition to the definition of the circuit the conductor width must be specified and also an estimate given for the final chip size. When the information is complete the input file and library file are compiled and an internal system file generated from which the layout program will proceed. Because several attempts are normally made to iterate to the best layout, this precompilation phase removes the need for the data to be re-checked for every iteration.

The program builds a chip layout in a series of successive levels working from one edge (bottom) of the chip progressively towards the opposite edge. At each level, an area (slot) at a time is locally optimized; components are chosen for each slot which are most closely connected to the existing portion of the layout and to any other components already selected for the current slot. Placement of components and routing of conductors takes place in parallel allowing a global view of the problem to be used and making it possible to organize the layout very efficiently.

The program begins by taking the estimate of the chip size and assuming a square chip, calculates the side length of the chip. The bottom edge of the chip is then laid down and one of the groups of pads specified in the input file is assigned to this side. The set of pads is allocated by finding the component most closely connected to all the pads then choosing the set of pads most closely connected to this component. This means the first group of pads will, as near as possible, all have a target connection in the first slot thus reducing the amount of routing to successively higher slots until the connection is made.

Once the first group of pads has been allocated, the other groups of pads are assigned to other sides of the chip in the order specified in the input language.

Layout in earnest can now begin. The layout proceeds from this point in a series of rectangular slots, each defined by the lowest level reached in the already completed layout and limited horizontally by obstacles protruding up at either end of the horizontal plateau. (see Figure 2.) Obviously the

```
LIBRARY FILE:
"COMP" INV(7,63)
    "PINS"
     1(0,14)M
     2(0,24)M
     3(4,63)M
     4(5,0)M;
  "RECT"(1) 0,0:7,12;
  "RECT"(1) 0,15:7,13
  "RECT"(1) 0,52:7,11;
 "GROUP"H (3,4) 1,1,0;
  "RECT"(3) 2,11:3,5;
 "GROUP"H (3,4) 1,22,0;
  "RECT"(3) 2,23:3,30;
 "GROUP"H (3,4) 1,55,0;
  "RECT"(5) 0,21:7,7;
  "RECT"(5) 0,10:7,7;
    "END"

"COMP" NAND(7,70)
    "POWER" 2
    "PINS"
     4(4,70)M
     3(7,36)M
     1(7,27)M
     2(7,16)M
     5(4,0)M;
  "RECT"(1) 3,3:8,12;
  "RECT"(1)3,18:8,7;
  "RECT"(1) 3,28:8,12;
  "RECT"(1) 3,63:7,10;
 "GROUP"H (3,4) 4,4,0;
  "RECT"(3) 4,14:6,5;
  "RECT"(3) 4,24:6,5;
 "GROUP"H (3,4) 4,34,0;
  "RECT"(3) 5,35:3,29;
 "GROUP"H (3,4) 4,67,0;
  "RECT"(5) 3,66:7,7;
  "RECT"(5) 3,3:8,7;
  "RECT"(5) 3,13:8,7;
  "RECT"(5) 3,23:8,7;
  "RECT"(5) 3,33:8,7;
    "END"

SOURCE FILE:

   "UTIL" 400
  "CONWID" 4
"LIBCOMP" INV1(INV)
"LIBCOMP" INV2(INV)
"LIBCOMP" NAND1(NAND)
   "PADGROUP"
 "LIBPAD" PAD1(PAD)
"LIBPAD" GROUND(PAD)
    "END"
   "PADGROUP"
 "LIBPAD' POWER(PAD)
    "END"
"NODE" POWER,INV1(1),INV2(1),NAND1(1);
"NODE" GROUND,INV2(2),INV1(2),NAND1(2);
  "NODE" PAD1,INV1(3),INV2(4);
  "NODE" INV2(3),NAND1(5);
 "UNCON" INV1(4),NAND1(3);
   "FINISH"
```

Figure 1

first slot is the width of the entire chip and successive slots are generated in much the same way that rectangular stones of varying size are laid when building a stone wall.

An attempt is made to optimize the use of the area in the slot both by choosing the best set of components to fit the area of the slot and by minimizing the amount of interconnection required. In filling the slot with components it is possible to calculate the total width required for a given set of components. This is simply the sum of the widths of the components (with appropriate orientations) and the width required for routing conductors vertically between the components. This latter factor consists of the space required for conductors going straight up through the slot without connecting to anything and the space required for connections to side pins of components. If the order and orientation of the components is well chosen this space requried for vertical routing can be minimized. For instance if a net connects to two or more pins in the slot there are two possibilities; (1) the pins are in different intercomponent gaps, in which case two or more vertical conductors are required and (2) the pins are in the same gap, in which case only one vertical conductor is required to service both of them (see Figure 3).

There are several other features which influence the orientation and ordering of components in a slot. Firstly, it is more efficient, in general, to have components oriented with their largest sides horizontal. This tends to produce a few large plateaus on which to build further slots rather than several small ones thus increasing effective use of area due to reduced fragmentation. Secondly the orientation of components is a critical factor in determining the number of conductor cross-overs required. For electrical reasons it is important to have as few cross-overs as possible so, within the constraints set by the previous requirements the components is a critical factor in determining the number of cross-overs in their connections with the already existing part of the layout. The third factor on which placement depends is the ordering of the components in the slot.

An attempt is made to minimize the horizontal routing required in the slot by taking this into consideration when ordering and orientating the components. There is obviously a trade-off between reducing the space requried for vertical routing and reducing the amount of horizontal routing since

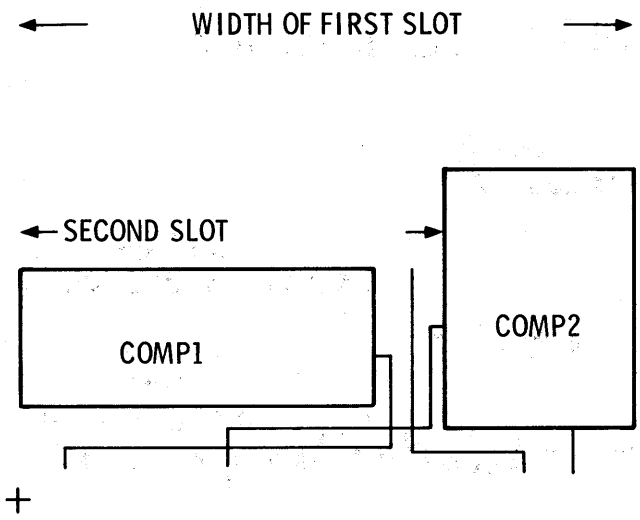


Figure 2

in order to put two pins in the same intercomponent gap it may be necessary to move one of the components further away from its best horizontal position and increase the amount of horizontal routing required.

These processes will have reduced the amount of space required for the current set of components as far as possible. If there is some extra space left in the slot an attempt is now made to find the component which will best use that space. When this operation has been completed the slot will usually be very tightly packed with little unused space. However any unused space is now inserted next to the lowest component in the slot in order to increase the width of what will probably turn out to be the next slot. The x coordinates of the components are now fixed allowing the x coordinates of connection targets to be fixed so that horizontal routing can take place. The y coordinates of the components are left undefined and are only fixed as each component has all routes which need to pass beneath it completed. The component can then be allowed to settle on top of the horizontal routing under it. This delayed fixing of y coordinates is necessary because it is extremely difficult to calculate the amount of space required for routing under a component

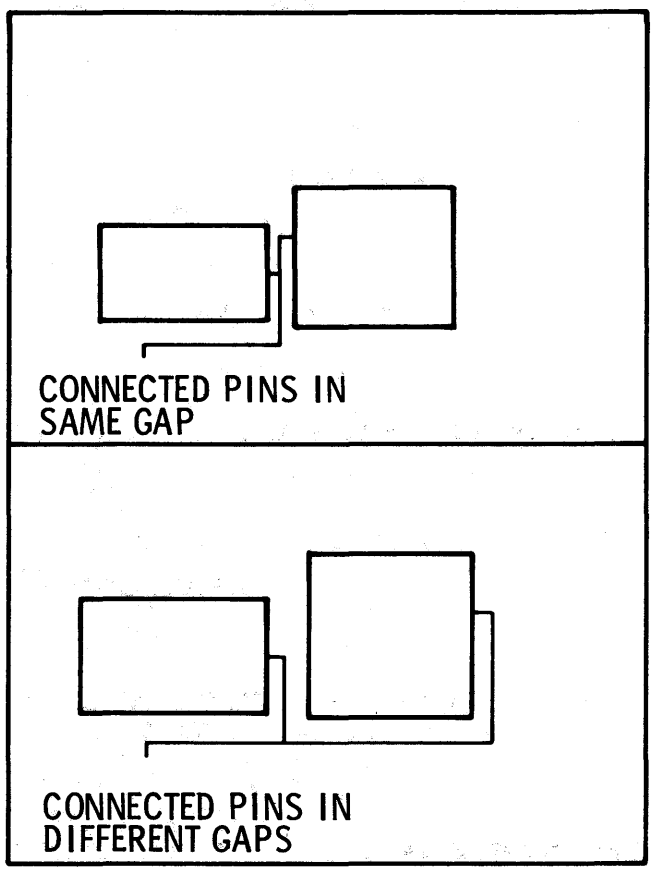


Figure 3

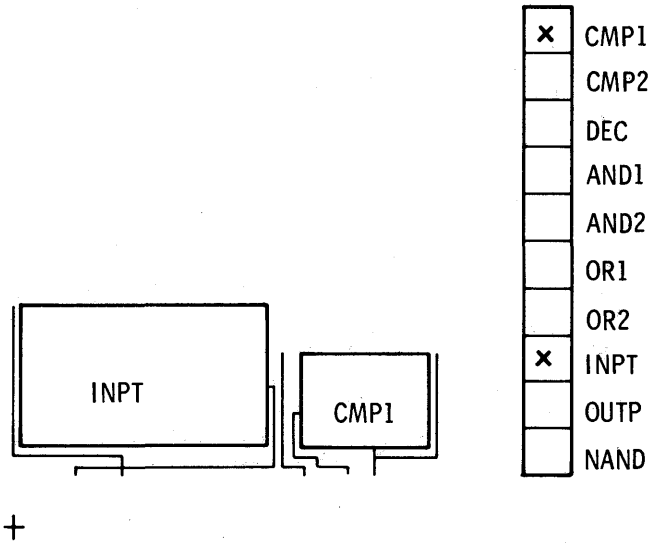


Figure 4

without actually doing it. The next part of the operation is to perform the horizontal routing in the slot. Because all the coordinates have been fixed it is possible to route horizontal segments even though the exact positions of the components are not known. For this routing a system of priorities is used. If there are two or more types entering the slot from below that need to be connected, these are given highest priority so that they are connected as early as possible and do not interfere with the actual intercomponent routing. Next highest priority routes are those which need to pass under components. The effect of completing these is that the components may be placed as low as possible in the slot and take less height. Lowest priority routes are those that only pass from the bottom of the slot to the top, and routes from component sides to the top of the slot. The algorithm operates by performing horizontal routing at increments of one conductor pitch from the bottom of the slot. As many routes and potential routes are completed at each routing level and when no further routing is possible vertical connections are made from the lower ends up to the next level and a fresh attempt is made there. As all routes below each component are completed, the component position is fixed and vertical routing to it is made. In parallel with this activity routes going to the top of the slot for connection in later slots are made and when all the components are fixed and the partial routing completed the processing of the slot is finished. This process continues for all the slots until eventually there will be no more components left to place and all the loose conductor ends will be joined in the final slot. As soon as the layout is finished all the pads are added and an output file is produced containing the details of the layout.

This fully automatic system, when tested on a manually implemented design produced a chip about 1.25 times the linear dimension of the manual design. It was found that there were several instances of awkwardly placed components or densely connected components where the program did not produce an optimum layout. In order to improve the

situation an interactive phase has been added which allows the program and the user to work together. A menu containing the names of all the components in the design is displayed on the right hand side of the display screen on which the design is drawn (see Figure 4). At the beginning of each slot the user is given the chance to choose the component positions and orientations. He may either fill the slot or break off on a partially full slot and instruct the program to fill the rest of the slot automatically or to leave the rest of the slot vacant. Using this facility the program improved on the completely automatic technique giving a result about 1.15 times the linear dimension of the manual layout.

It is worth pointing out the reasons the program does not equal or improve upon the layout produced manually. There are two main reasons. The main one is that the manual designer often designs his components to butt together. The program is unable to detect this fact and will always leave space for, and perform, some routing between the components. Research is under way to test the feasibility of providing some method to butt together some components automatically. In a completely "standard cell"[1] system a different algorithm is used which takes advantage of all the aspects of "standard cells". The second reason why the automatic layout is larger is that designers often generate special versions of modules to fit into local contexts. This has the effect of reducing the amount of connection required had the "standard" version been used. The program, having no knowledge of what is contained within components, is unable to adjust component sizes or shapes and must make do with what is provided. For small numbers of custom designed IC's the design time/chip size trade-off is a good one and will become better for even larger numbers of production units when the new VLSI processes become commonplace and space ceases to be critical.

POST PROCESSORS

The interactive design program produces technology independent output assuming there are two layers available for routing. In order to realize the circuit in the required technology an appropriate post processor is required. Typically the post processor takes the design file produced by the layout program and assigns conductor segments to layers, inserts underpasses where necessary and locates contact windows where interlayer connections are required. It is at this post processor stage that we need to ask the question "What is going to happen to the design from here?" At Edinburgh the design is passed to an independant suite of drafting programs collectively known as GAELIC.[2] The post processor searches the source and library files for the component definitions and extracts from them the details of the internal logic specified in GAELIC manual input language. This is output to a file together with all the interconnections and contact windows in the GAELIC manual input language, thus presenting the GAELIC system with a complete design which looks as though it were produced manually. Using special programs in the GAELIC suite it is possible to automatically generate drive tapes for pattern generators and other mask-making devices. The use of the MAGIC suite of programs in conjunction with GAELIC substantially reduces the time required to design an integrated circuit. Used properly, the system can reduce design time by a factor of ten. This is achieved by storing in the library the complete component specification from the previous circuits; as the library is built up, all that is necessary to perform a new design is to select appropriate items from the library and to specify the interconnections. After this stage everything can be automatic. The chip is laid out by MAGIC and the appropriate GAELIC post processor is used to generate drive tapes for a mask making device. If any interaction is required, the facilities exist, not only in MAGIC but also in terms of the substantial graphic editing facilities in GAELIC.

APPLICATION CONTEXT

The MAGIC suite of programs is applicable to all areas of IC design in which modularity of design exists. Both the rectangular components described earlier and standard cells can be laid out, although the latter layout is not as efficient as if a specialized standard cell program had been used.

The system is most effective when used for irregular designs where patterns are not repeated a larger number of times such as in memory chip designs. In the latter instance, the designer can often produce an optimal circuit design in such a short time that the overhead represented by the extra space requirement using MAGIC is not warranted. The system is applicable to all current IC technologies since sufficient parametrization exists for the details of individual technologies to be included in the design specification. The use of post processors, one for each technology, allows the design to be optimized for each particular technology automatically.

As VLSI techniques become more commonplace the difficulty of laying out a chip by hand is going to be so extreme that unless systems like MAGIC are used, the full potential of VLSI is not going to be realized.

REFERENCES

1. Schweiker et al., "The LTX System," 1976 Design Automation Conference.
2. Eades, J., "The Design of an Interactive Computer System for Microelectronic Mask Making," Ph.D. Thesis, University of Edinburgh 1976.

STICKS—A graphical compiler for high level LSI design

by JOHN D. WILLIAMS
Hewlett-Packard Data Systems IC Laboratory
Cupertino, California

INTRODUCTION

The need for good design automation in the area of integrated circuit layout is severe. It is believed that the STICKS system does much to fill that need. STICKS is a computer aided design system which frees the designer from the tedious aspects of IC design and allows him to concentrate on the more creative and necessarily human side of the design process. With STICKS the designer is allowed to divorce himself from the usual precise sizes, spacings, and interrelationships required for an IC layout and, instead, submit his creativity in the form of rough, freehand sketches. The computer makes whatever adjustments are necessary in order to generate an error-free layout. The opportunity is then given to verify and, if necessary, modify the computer's interpretation of the design. An interactive loop is thus formed with the computer doing all of the tedious, detailed work and the human designer providing the necessary inspiration.

Using traditional methods, large scale integrated circuit layout is a tedious, time consuming and error prone process. IC layout is a procedure which results in the placement and interconnection of each of the thousands of components which form the integrated circuit. The designer accomplishes this by drawing several superimposed "mask" layers using a drafting board which together, control the areas where various chemical diffusions and etchings will be made on the silicon wafer that forms the IC. Components such as transistors, diodes, resistors, and their interconnections are formed by various combinations of these masked areas. Unfortunately, the specific sizes of and spacings between components is critical, preventing the designer's job from being an easy one.

There is an overall goal in IC design to pack as much circuitry as possible into a small area. The more circuitry a particular IC contains, the more valuable it is and, consequently, the more marketable it is. The physical size of the chip, however, is a major factor constraining the number of components. Due to a parameter known as defect density, the larger the chip, the less likely it is to work and the more it will cost. Defect density is the number of random material defects per given area which result both from the starting silicon material and from normal losses in the manufacturing process. The solution, one would think, would be to simply make everything smaller thereby increasing component density. Other limitations in the capabilities of the process, however, make this solution impossible. As the components get smaller, the more effect small random defects will have and, again, the less likely the circuits will work.

The absolute minimum sizes and spacings in a layout are embodied in a set of "design rules" which are specific to the particular manufacturing process. They specify in detail the relationships between any two components or, more fundamentally, mask level rectangles. Only when all of the design rules have been met will the circuit be guaranteed to be manufacturable.

It should be understood that the design rules are not necessarily simple width and spacing definitions, they are often dependent upon complex relationships between the mask rectangles and the components intended to be implemented. The number and complexity of the design rules are a major contributor to the difficulties and tedium encountered in the layout process. Each mask rectangle is constrained in two dimensions by several other rectangles on the various mask levels. Indirectly, the placement of every rectangle in every mask is dependent upon the placement of most, if not all, of the rectangles in the entire IC.

The designer approached the generation of the layout by placing the first element and then, sequentially, building elements around it, while trying to leave just enough room for future elements. This is fairly difficult because it is hard to know just how future elements will fit together. The designer generally copes with this problem by adopting a trial and error approach. When the rectangle he is trying to place will not fit without violating a design rule, he must backtrack far enough to correct the problem. This is accomplished by erasing and repositioning any previously placed rectangles which led to the difficulty. Figure 1 shows a simple example of this situation. This particular example is not a serious one but, in actual practice, several hundred structures may have to be erased and redrawn each time a problem of this type occurs. It is virtually impossible for even a highly experienced designer to be able to look ahead far enough in order to avoid having to backtrack.

Traditional layout methods pose additional problems besides those already mentioned. The number and complexity of the design rules make a certain number of design rule violations inevitable. In fact, we have found that approximately ten undetected design rule violations are made for every thousand hand drawn rectangles. This number rep-

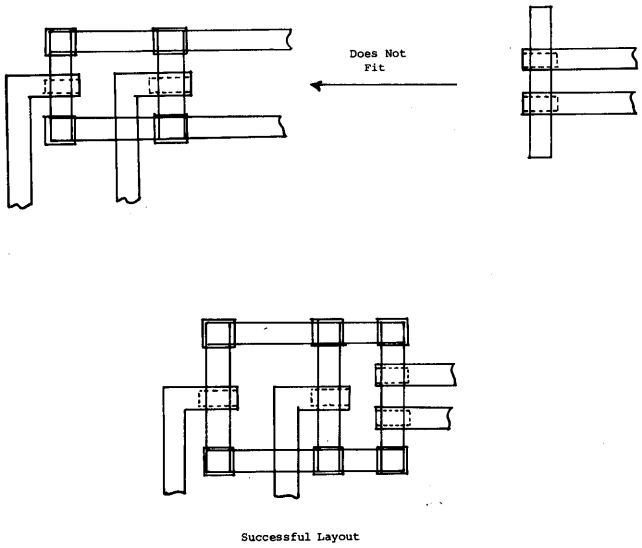


Figure 1

resents only those errors which are found after the layout has been completed. These can have disastrous effects if global design changes must be made to correct them. The problem is that the designer has packed all of the mask rectangles as close as possible together (within design rule constraints) in an effort to maximize circuit density. If a design rule violation is found, it means that the structures were packed too closely together, implying that they must be moved apart slightly. Unfortunately, moving these structures apart would cause additional design rule violations with surrounding structures. The corrections for these errors tend to ripple out to the edge of the drawing, forcing the repositioning of many structures. Since the errors were found only after the entire layout had been completed, thousands of structures could be involved. Then, inevitably, the process of physically repositioning all of these elements could easily produce more design rule violations causing the problem to repeat.

The need to reposition large numbers of circuit elements can also occur when actual circuit modifications must be made such as when logic errors are found after the IC has been fabricated for the first time. This has the same effect as the discovery of a design rule error. Once the layout has been created, it is often very difficult to add any additional circuit elements without moving a lot of rectangles. Again, changes like this are always fraught with the danger of introducing new errors.

With these difficulties in mind, it is easy to understand why layout represents a very costly stage in IC development, both in terms of time and effort.

THE STICK DIAGRAM

The designer's input to the STICKS system, called a stick diagram, is a high level, schematic-like representation of the intended circuit. Circuit elements like transistors, diodes, and contacts (vias) are represented as single symbols instead of as complex sets of overlapping mask levels. Likewise, silicon and metal interconnects are represented by their centerlines. A stick diagram is unlike a schematic, however, in that it includes rough relative positional information in addition to the usual connectivity information. An example of a stick diagram is shown in Figure 2. Like a schematic, the actual distances between the elements in this drawing are completely arbitrary. What is important however, is the relative position between the elements. The fact that a transistor, for example, is "above" a metal interconnect or "somewhere to the left of" a particular diode are the properties which make a given stick diagram unique.

The lack of actual geometrical spacing information is an important characteristic of the stick diagram. In using the stick diagram as a design "language", the IC designer need not worry about the actual location of the circuit elements he places. As a result, his design is nearly as easy to draw, modify, and interpret as a schematic. In fact, it has been found that many designers bypass the schematic entirely and directly translate their logic diagrams into stick diagrams. The use of single symbols for complex structures and centerlines for interconnects as well as the complete absence of specific inter-element spacings makes the stick diagram very fast and easy to draw. The designer does not have to worry about leaving enough room for surrounding structures, there is always room to add them. Unlike designs produced at the mask rectangle level, modifications can be made on a stick drawing without the need for accompanying major global rework. If it is necessary to add a structure, one can always "find" room by simply drawing it smaller and inserting it where desired. Likewise, deleting a structure does not leave a hole since the spacing between elements in a stick drawing is arbitrary.

This arbitrary spacing also means that the designer is removed from the specifics of the particular manufacturing process. He does not have to know what design rules are or where they are applied, nor does he have to know that mask levels like nitride, implant, diffusion, and oversize gate even exist (e.g., a transistor is a single symbol instead of a complex system of precisely aligned, overlapping masks).

The difference between a schematic and a stick diagram (the addition of relative positional information) makes the translation of the stick diagram to a full mask layout a fairly straightforward process. Without this added information,

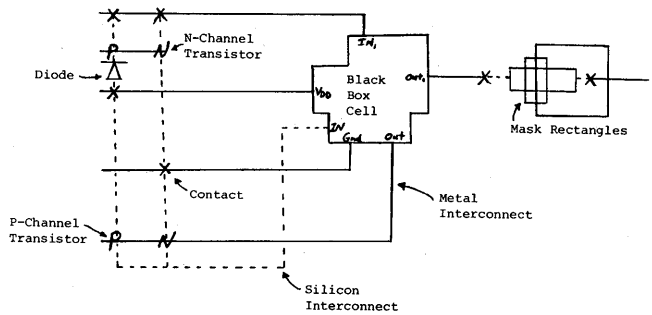


Figure 2

the translation process would have to include component placement; a problem with which computers traditionally have had a difficult time.

The stick diagram is a happy medium between the schematic and the full mask layout. It is at a high enough level to allow the designer to concentrate primarily on the creative aspects of the design yet at a low enough level to completely describe the final geometries in such a way that translation to full mask rectangles can be automated without loss in layout quality.

IC design at the stick diagram level utilizes the capabilities of both man and machine in a very effective way. Humans have very good global perspective; they can look at a drawing and immediately identify how elements should roughly fit together to maximize density. They are very poor, however, at identifying all of the constraints on each element of the drawing and precisely aligning them so that no design rules are violated. The computer, on the other hand, is extremely good at calculating local element separations but poor at making global decisions. By allowing the human to design with stick diagrams and having the machine translate these drawings into full layouts, each is doing what he does best. The human is using his experience and global insight to create the overall plan for the layout while the computer is left to do the tedious, detailed work that it does so well.

In addition to transistors, contacts, and interconnects, the stick diagram may also include mask rectangles and cells (previously designed layouts).

There are two different types of cells available to the designer. The first type, called a "macro," is simply a stick diagram which has been previously stored in a library for future use. When a macro is placed in a stick diagram, it acts as though it had been drawn by hand in its assigned location. It loses any identity it may have once had and, upon subsequent translation to mask layout, is free to stretch, shrink, and mold to its environment depending upon the influenc of surrounding structures. The value of macro cells lies in the ability to pre-define small, commonly used sub-circuits such as gates and flip-flops, eliminating the need for the designer to repeatedly draw them.

The other kind of cell is called a black box cell. It is created by allowing STICKS to generate a layout from a stick drawing (which may, incidentally, contain other black box cells) and then freezing this layout by declaring it to be a new black box cell. This cell may be used in subsequent stick drawings as if it were a logic symbol. It is an outline of the actual layout with labelled ports signifying places where external circuitry may be connected. The cell is also labelled to denote its function. The use of black box cells allows the designer to operate at a higher level than that of transitors and diodes. He can define functional blocks of circuitry and use these in future stick diagrams.

THE STICKS COMPILER

The job of the STICKS compiler is to translate the designer's stick diagram into a complete mask layout which is consistent with design rules. It performs this operation in two main stages. The first stage (called the spacing algorithm) repositions each of the elements in the stick diagram in a way which brings them together as close as possible without violating design rules. The second stage (called rectangle generation) converts the resulting "spaced" stick diagram into a full mask layout.

The spacing algorithm's ability to function properly is due to the fundamental nature of the stick diagram. The stick diagram is composed of relocatable elements wired together by stretchable interconnects. It is this stretchability which provides the added degrees of freedom that are necessary to create an error free design. As long as each element is free to move and any connections to any other elements are infinitely stretchable, a valid layout can be produced. This is true because of the way IC design rules are defined. Design rules prevent any two elements from being too close to each other; all design rules are specified as a minimum distance. This implies that a perfectly valid layout could be produced from a stick diagram with the above properties simply by stretching all of the interconnects so that elements are infinitely far apart from each other. Opposing this philosophy, however, is the desire to maintain the highest possible layout density. The solution then, is to move the elements only as far apart as necessary in order to not violate any design rules. This, in theory, will create the smallest possible valid layout.

The problem, then, is to know in which direction and how far to move each element to produce the optimum layout. The approach the STICKS compiler takes is to divide this two-dimensional problem into two one-dimensional problems. This is accomplished by treating the horizontal structures separately from the vertical structures. In doing so, the algorithm only has to worry about moving each element in one direction at a time. Basically, the spacing algorithm begins by placing all structures which lie on the bottommost horizontal coordinate at the bottom (origin) of a blank work area. The next step is to attempt to place the structures which are on the second horizontal coordinate as close as possible to this origin. This is accomplished by looking up the design rules between the various structures on the two coordinates and deciding where each second coordinate structure must be in order to be as close as possible to the first coordinate structures without violating design rules. The second coordinate structures are then actually moved to these new locations and assigned an actual distance from the origin. The compiler has done two things: it has moved structures relative to each other in a way which tends to increase circuit density and it has also assigned an actual physical vertical spacing between structures.

It should be realized that whenever a horizontal structure is moved to a different coordinate, any vertical interconnections which are attached must be stretched accordingly. This maintains the connectivity of the circuit.

The horizontal pass continues with each subsequent coordinate of horizontal structures moved down as far as design rules permit. When all of the horizontal structures have been spaced, the spacing algorithm is used in the opposite direction. In this pass, the vertical structures are reposi-

tioned while any horizontal interconnects are appropriately stretched.

As soon as both passes have been completed, the designer is given the ability to perform optimization on the stick diagram with the help of the compiler. These techniques will be discussed briefly in the next section. Once the designer is satisfied with the density of the spaced stick diagram, he can ask the compiler to go into its second stage of operation: rectangle generation.

As its name implies, rectangle generation takes the stick diagram which has been spaced to design rule specifications and builds the corresponding mask rectangles. It does this by expanding the interconnect centerlines to their full width and by generating all of the mask rectangles associated with the circuit elements. It also recalls the contents of black box cells from the system library and substitutes the appropriate set of mask rectangles for the cell outline found in the stick diagram. There is no danger of a design rule violation occurring as a result of this procedure since the spacing algorithm took all of this into account when it performed its operation. The product of rectangle generation is a complete mask layout ready to be sent to mask making equipment. Figure 3 shows the results of each stage of the compilation of a simple stick diagram.

One section of the compiler (called the knowledge base) consists of a set of production rules in which are coded the design rules between any two structures. Whenever the compiler is trying to decide how far apart two given structures must be, it consults its knowledge base which, in turn, analyzes the properties of the two structures and reports back the minimum separation distance. Besides the basic design rules, the production rules include additional knowledge which might be called layout heuristics. Layout heuristics are the designer's "tricks of the trade" which he has developed through experience. One example of this would be the fact that a design rule which specifies the minimum separation between two metal interconnects can be ignored if the two structures are electrically connected. In fact, it might be advantageous to completely merge these two structures thus saving area. STICKS has the capability to identify situations like this and make appropriate decisions.

There are several advantages in basing the implementation of the STICKS compiler on the use of production rules. First, the complexity and tremendous number of relationships between IC structures make it very difficult to develop a traditional algorithm which is capable of performing the required tasks. The IC design rules as well as layout heuristics already exist in the designer's mind as a set of rules which are selectively applied whenever the proper circumstances exist. It is natural then, for a computer program to incorporate the same set of rules as its knowledge base. A second advantage is that it is impossible to incorporate all knowledge of IC layout the first time. In fact, it is almost certain that a continuous process of "tweaking" the performance of the system by modifying the knowledge base will exist throughout its usable life. Having to continually rewrite or patch an algorithm is not a pleasant prospect. Modifying a production rule based system, however, means only adding, deleting, or modifying the appropriate rules. Since the rules are virtually independent of each other, this can be done without adverse consequences.

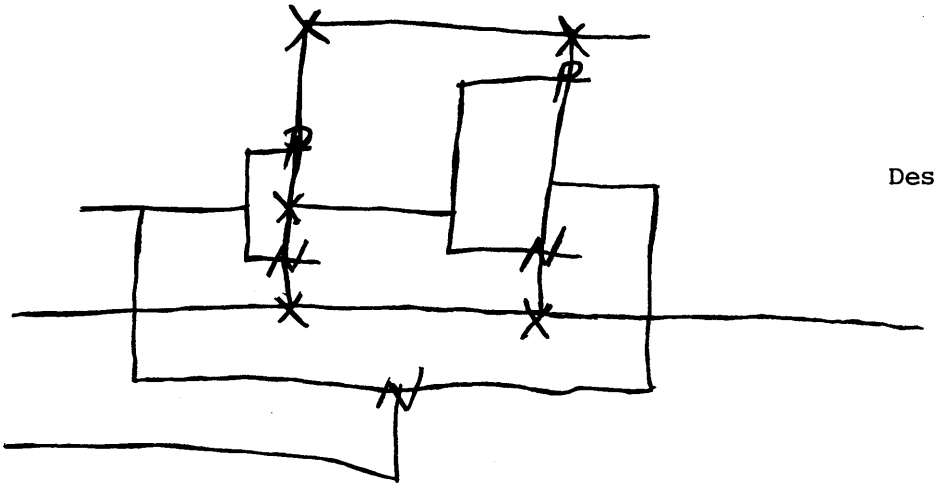
DESIGNER INTERACTION WITH STICKS

An important feature of the STICKS system is that the designer has total control over the quality of the final layout. This control is maintained through several interactive functions which may be invoked after the compiler has performed its spacing algorithm on the entire drawing in both the horizontal and vertical directions. At this point, there are no design rule violations and all structures have been packed toward the lower left corner of the drawing.

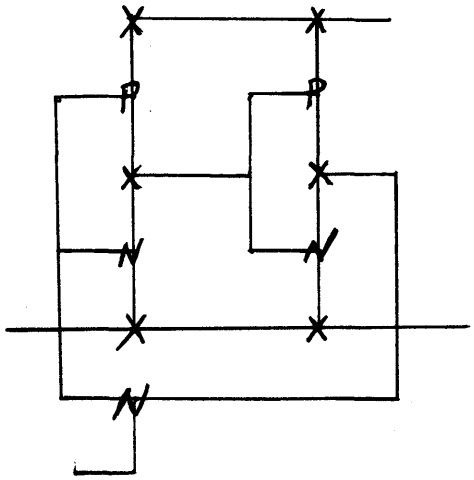
It is likely that this spaced stick diagram will not be as dense as it could be. Although each structure in the drawing will have been positioned as close to its adjacent structures as design rules permit, it is quite possible that unused areas exist. This stems from the fact that the spacing algorithm really is not looking at the global picture. It is assigning minimum spacings between adjacent components but cannot identify the situations when overall density might be improved if a particular local spacing were not set to minimum. This occurs most frequently when compression in the horizontal direction blocks subsequent compression in the vertical direction. Figure 4 shows a very simple example of this situation. In this example, a much more compact layout would have been obtained if the first group of horizontal structures in the center of the drawing had not been moved all the way down to the origin but, instead had been left in its original location. The compiler had no way of knowing this, since it just tries to pack everything toward the bottom left of the drawing. STICKS depends upon the designer to spot problem situations like this and allows him to remedy them through the use of a procedure called local optimization.

Local optimization allows the designer to draw a boundary around any area in the spaced stick diagram and specify a direction in which structures that are included in the boundary are to be respaced. Figure 5 shows how this operation would work to fix the problem in Figure 4. It is done in two steps. First the horizontal group of structures which were originally moved down to the origin are directed to move back up and then the displaced vertical structures are directed to move to the left to fill in the unused area. There is no danger of design rule violations occurring as a result of this procedure since the spacing algorithm always does the actual repositioning of structures. The designer is merely giving it guidance from his global considerations.

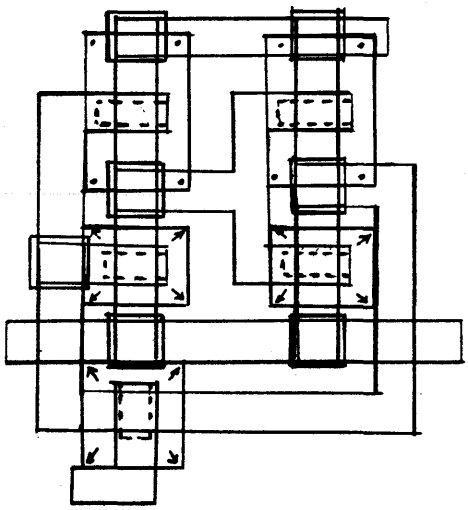
A second means by which the designer may interact with the system is through the use of fractures. Figure 6 is a before and after picture showing how fractures can be used to increase the density of a spaced stick drawing. The overall effect of fractures is to allow sections of circuitry which were originally constrained to a particular location by straight interconnects to fill in large unused areas of the layout. The fracture breaks a straight interconnect and allows it to jog in either direction at that point. The designer is actually giving his drawing an added degree of flexibility



Designer Input



Output of Spacing
Algorithm
(Spaced Stick Diagram)



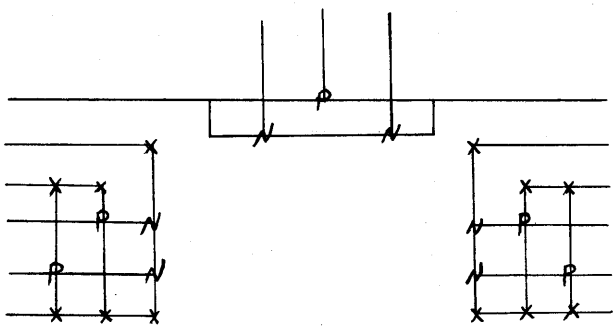
Output of Rectangle
Generation
(Full Mask Layout)

Figure 3

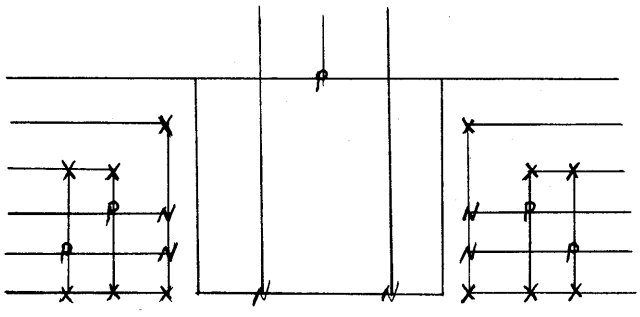
in certain select areas, thereby giving the STICKS compiler freedom to do a better job.

Another problem which can occur involves groups of circuit elements which the designer would like to physically keep together in the layout. It is desirable, for example, to keep all of the flip-flops in a register together in a single row. This is especially important for circuit debugging purposes in the event that the output state of the register must be probed. It would be awkward to search all over the IC for all of the pieces of the register in order to identify its output nodes. This, however, might very well be the case if the spacing algorithm were left unchecked. It has no way of knowing which functional groups of elements should remain together. To prevent the dispersion of such functional groups, the designer is given the ability to enclose all of the elements of the group in a rectangular perimeter called a corral. The corral has the ability to change absolute size; however nothing which is enclosed in it may leave its boundary. This effectively separates those structures from all others in the drawing.

The final way in which the designer may interact with the system is by specifying the absolute micron distance between any two structures in the drawing. This effectively limits the freedom which STICKS has to move structures around, giving the designer considerable control over the

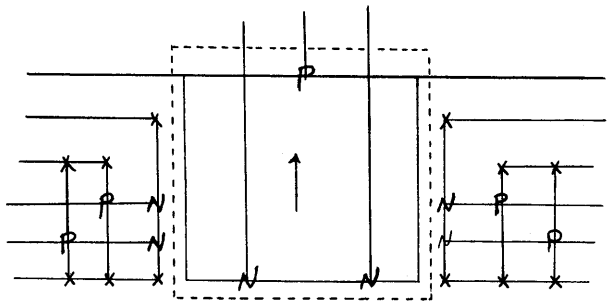


Original Stick Diagram

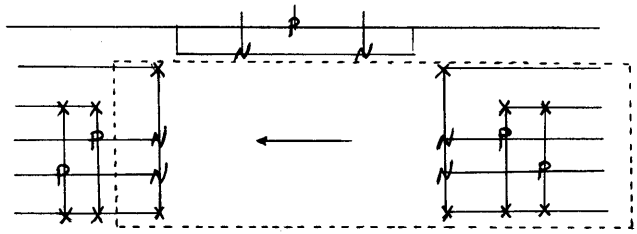


Horizontal Downward Movement Blocks
Compression in Opposite Direction

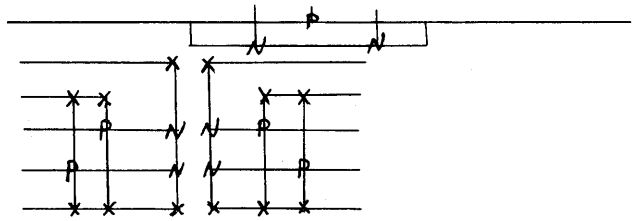
Figure 4



Upward Local Optimization



Local Optimization To Left



Stick Diagram After Designer Interaction

Figure 5

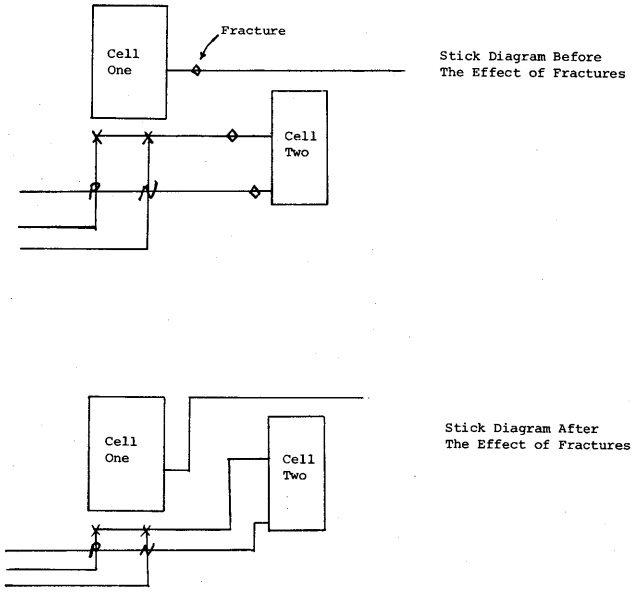


Figure 6

final shape and size of the layout. Of course, there are perils associated with this ability. It is easy to over-constrain a drawing, especially when an absolute constraint imposed by the designer violates design rules. STICKS can identify these situations and inform the designer of the impossibility of his restrictions.

Together, these interactive optimization techniques allow the designer to maintain ultimate control over the quality of his design. This control is believed to be essential in a design automation system of this kind.

CONCLUSION

The STICKS system offers the integrated circuit designer many advantages over traditional layout methods. He designs in a high level, graphical language which is very easy to draw, modify, and maintain. He no longer must worry about design rule violations since it is impossible for him to make any. The layout is built to design rule specifications.

Additionally, the stick diagram possesses a certain amount of process independence. This means that once an IC is designed in the form of a set of stick diagrams, it may be "recompiled" into different mask layouts which are compatible with other manufacturing processes. This feature helps to prevent the obsolescence of expensive IC designs. It is especially advantageous as the original process evolves. The designs can track the process by being recompiled using each new set of design rules.

A final advantage is related to the ease with which the designer can produce a layout. Traditional layout methods are so tedious that even when it is discovered that the layout is poor with respect to circuit density, it is often just too much trouble to do anything about it. With STICKS, on the other hand, the designer is encouraged to try several different layout approaches and try to optimize for density. As a result, it is quite possible to obtain even higher quality layouts with STICKS than with a layout done completely by hand.

Together, these advantages make STICKS an exciting solution to the difficult IC mask layout problem.

REFERENCE

1. Williams, John D., "STICKS—A New Approach To LSI Design," Master's Thesis, Massachusetts Institute of Technology, June 1977.

A method for the automatic wiring of LSI chips

by NING NAN and MICHAEL FEUER
IBM Corporation
Hopewell Junction, New York

INTRODUCTION

The best known technique for finding a connection between two points is the Moore-Lee algorithm.[1] It is general and thorough, but leads to a long running time. Hightower[2] devised a line-probe wiring scheme which is much more efficient in finding such paths. Variants of this algorithm are probably the most popular means of wiring printed circuit boards at present. However, both of the above techniques suffer from a tendency for the last connections attempted to be blocked unnecessarily by previously wired ones. A more tentative approach to these connections was developed by Hitchcock,[3] whose cellular mazerunner specified connections through cells without fixing their exact location until the end. Schemes which we call line packers for assigning routes to parallel channels have been devised by Hashimoto and Stevens[4] and Kernighan, Schweikert and Persky.[5]

The method described in this paper was motivated by a similar desire to defer final channel selection as long as possible and to give each connnection equal treatment. The method consists of subdividing the wiring space into blocks in a hierarchical fashion, assigning connections to block boundaries by perturbing a global trial solution, mapping the solution to a finer grid and eventually allocating wire segments to channels in the vertical and horizontal directions by linepacking techniques.

The results of an implementation of this method, using a two level hierarchy, have proven very successful in production use at IBM.[6]

THE WIRING PROBLEM

The integrated circuit chip consists of a planar array of devices whose logic terminals (pins) are to be interconnected by metal according to a functional specification. The metal lines are made on two layers separated by an insulating material. In order to maximize wiring efficiency, the horizontal segments of a connection are made primarily on one plane (the horizontal plane) and the vertical segments primarily on the other (the vertical plane). The wiring channels on these planes are organized into horizontal streets and vertical avenues (Figure 1).

In this paper, a net is taken to mean a set of pins which are to be connected by metal wires or those wires themselves. A connection is a pair of pins which are to be interconnected, or the metal path between these pins. A segment is a portion of a net which exists within one street or avenue.

Changing planes can be accomplished by introducing via holes in the insulator between the two planes. These vias are made as needed and are not limited to the fixed patterns typical of printed circuit boards.

HIERARCHICAL GLOBAL WIRING

In the first phase of wiring, the chip is divided into a rectangular array of blocks each of which contains many wiring channels and circuit pins. A straight line segment separating adjacent blocks is called an edge. The function of global wiring is to decide which edges the global nets will cross. Nets whose pins are contained wholly within one block (local nets) are not routed at this phase. For large chips, each block can be subdivided after one stage of global wiring into smaller blocks, and the global solution can be further refined (Figure 2).

Each edge can accommodate a number of crossing wires. This is called the channel capacity of the edge and is largely determined by the number of wiring channels crossed by the edge. Each block is assigned an internal capacity reflecting the ease of wiring past the internal nets or other obstacles within the block. The goal of global wiring is to avoid wiring more nets through blocks or edges than permitted by the available capacities.

PERTUBRATION OF A TRIAL SOLUTION

The global wiring can be done, in principle, by any of the traditional mazerunning or line-probe techniques. An essential advance, however, was made by K. A. Chen[6] and later by J. Lee,[7] which we will call wiring by perturbation from a trial solution. The initial trial solution is built up by wiring each net indepenently. This results in an invalid solution where some regions are over-congested in the sense that wiring demand exceeds edge and block capacity. Nonetheless, this initial solution resembles the real solution much more closely than does a blank wiring image. In addition, the initial solution has the satisfying property of treating all the nets equally.

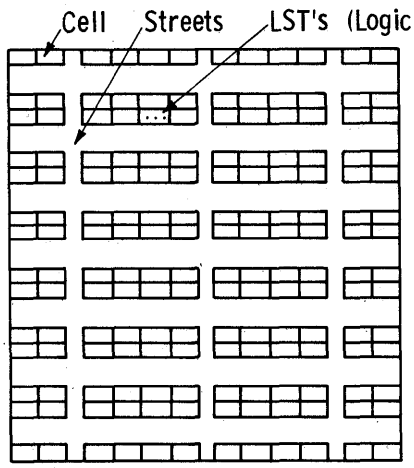


Figure 1—Wiring image

The net configurations, which can be used to build the initial solutiin, vary widely: minimum spanning trees, Steiner trees, biased Steiner trees, chains, source of sink trees, and redundant graphs, such as J. Lee's enclosing grid (Figure 3).

The perturbation consists of moving nets or segments away from congestion peaks. This rerouting can be accomplished by mazerunning, parallel translation of segments or by the deletion of redundant segments. In each case, a net or segment is identified as crossing a congested region and then rerouted in a less congested area at the possible expense of increasing its length.

An illustration of this global wiring procedure on an array of 3×3 blocks is given in Figures 4-8. Each number in Figure 4 represents a pin of a circuit. The same numbers on the chip are to be connected to form nets. All the nets are connected in a Steiner fashion without reference to each other. The initial global solution in Figure 5 is, in general, over congested at some boundaries. The number of nets crossing from one block to another is defined as the *channel demand* at the boundary between the two blocks (Figure 6). Where channel demand exceeds capacity, as outlined in Figure 7, there is clearly a need for rerouting some nets.

The reduction of congestion starts at the worst boundaries. Each net crossing the congested boundary is assigned a weight, which is a function of the congestion in the whole path of the net. The net with the highest weight is chosen first for rerouting. Since only improved paths are accepted in the process, the overall convergence is guaranteed. Figure 8 shows a balanced solution after rerouting.

The process can be visualized as the leveling of hilly terrain by pushing material from hill tops down the slopes. This results in lower and smoother contours with a small increase in total volume of material (net length).

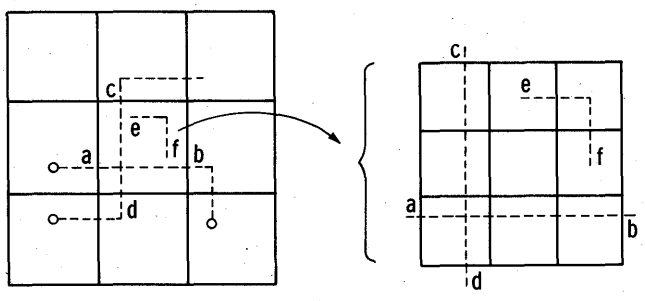


Figure 2—Block hierarchy

PIN SELECTION

During the last stages of global wiring or the early stages of detailed wiring, it is advantageous to reassign nets to pins. The pins to which the nets must attach may often be permuted in small sets without any effect on logical function. For example, the input pins of a NAND or a NOR circuit are all equivalent. The exact choice of these pins has an effect on the number of channels which are available through a block, as illustrated in Figure 9. The objective of the pin selection analysis is to minimize the blockage of the wiring sources.

NET SEGMENTATION

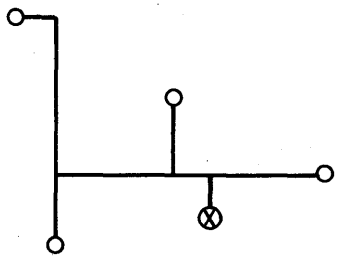
After global wiring, the nets are assigned to certain horizontal wiring streets and vertical avenues for detailed wiring. Each street will contain a number of parallel segments which must be assigned to available channels. The wiring within each street or avenue is dependent on that in all the others. In particular, where a street crosses an avenue, the vertical and horizontal wiring are coupled. Still, this coupling is generally not strong enough to force the detailed wiring to use more channels than would be required for an uncoupled solution.

Figure 10 illustrates the segmentation of a net N, in the vertical direction. X_i's are the net points and a_i's represent the assignment points. After the vertical assignment process, net N is segmented into four independent subnets n_1, n_2, n_3 and n_4.

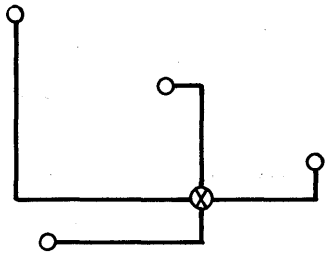
Figure 11 shows the horizontal segmentation of the same net N. It should be pointed out that only one segmentation is required for a wiring procedure.

CHANNEL ALLOCATION

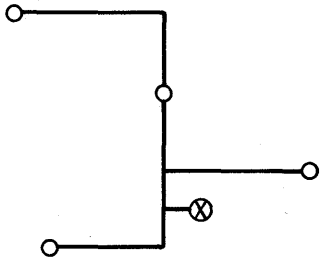
The most constrained streets or avenues are wired first. The problem consists of a set of parallel segments to be assigned to channels. The constraints come from two sources. The first is that there may be fixed target points (pins) within the street or avenue. The second is that a segment which enters a street, from one side on a specific channel must be wired nearer that side than a segment which enters the street from the other side on exactly the same channel. Constraints of the second kind are called *conflicts*.



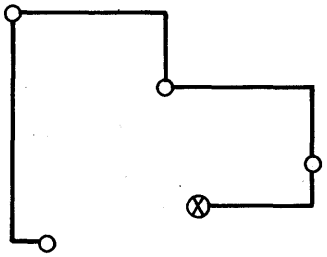
Steiner Tree (l=17)



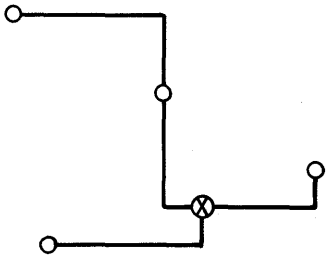
Source-Sink (l=23)



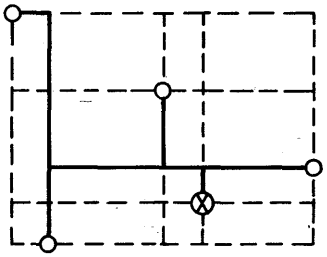
Vertically Biased
Steiner Tree (l=18)



Chain (l=23)



Minimum
Spanning Tree (l=19)



A Redundant Graph

Figure 3—Net configurations

The algorithms used for channel allocation are repeated linear assignment of segments to channels using Munkre's method[8] and an extension of the simple line pack procedures of Hashimoto and Stevens. The former is used when there are pins in the wiring street, the latter when the streets are clear. As mentioned earlier, some pin assignments can be deferred until the channel allocation phase.

An example of line packing in a horizontal street is given in Figures 12 through 15.

Net segments entering a street from above in the same channels used by other segments entering the street from below are in potential conflict. The net segment entering from above has to be assigned to a channel above the one assigned to the segment entering from below. This ordering

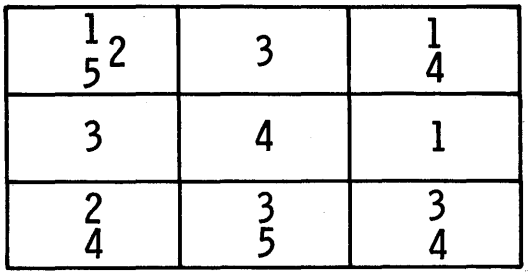


Figure 4—Sample global wiring problem

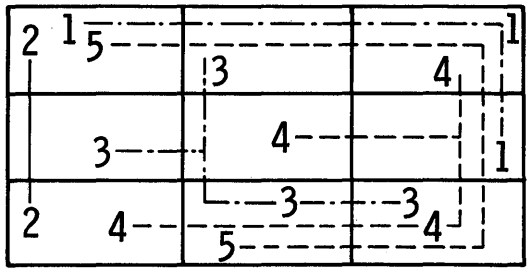


Figure 8—Global wiring after rerouting

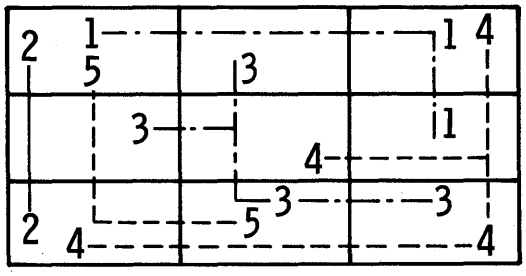


Figure 5—Initial global wiring

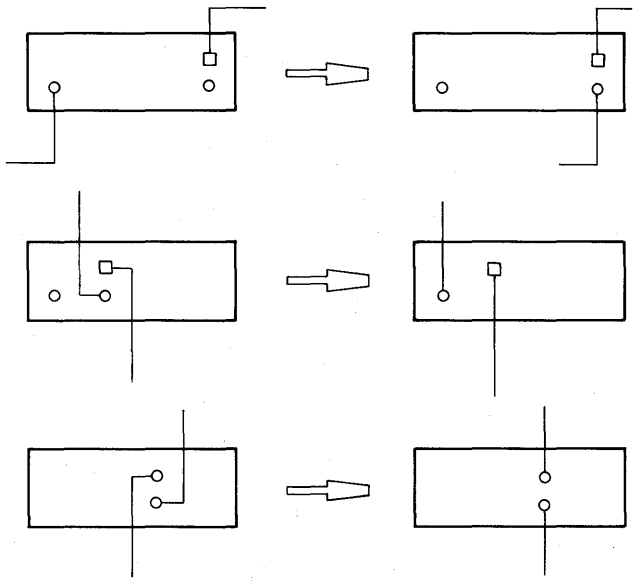


Figure 9—Pin selection analysis

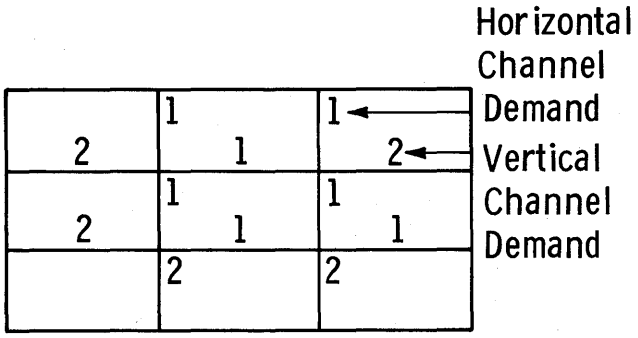


Figure 6—Channel demand

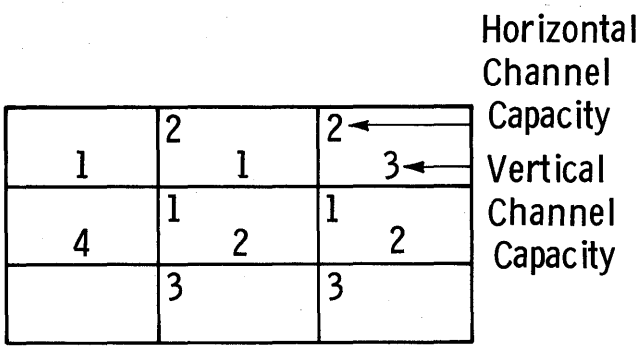


Figure 7—Channel capacity

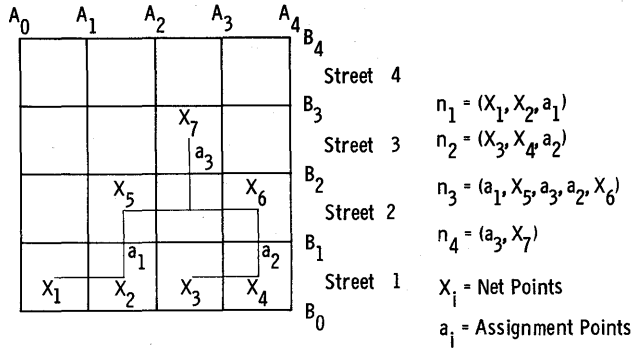


Figure 10—Vertical segmentation of net N

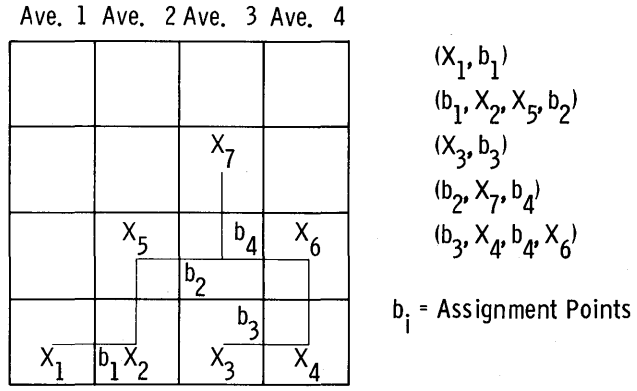


Figure 11—Horizontal segmentation

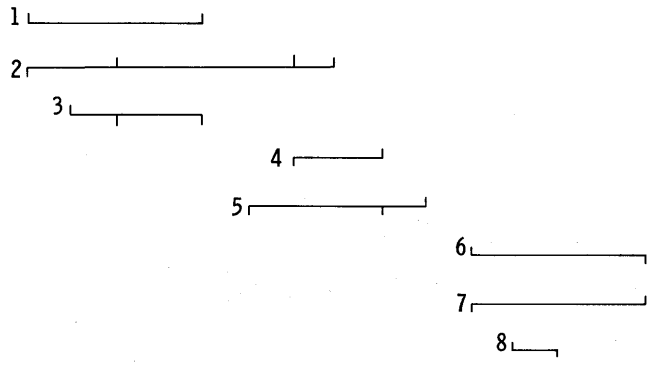


Figure 12—Net segments with conflicts

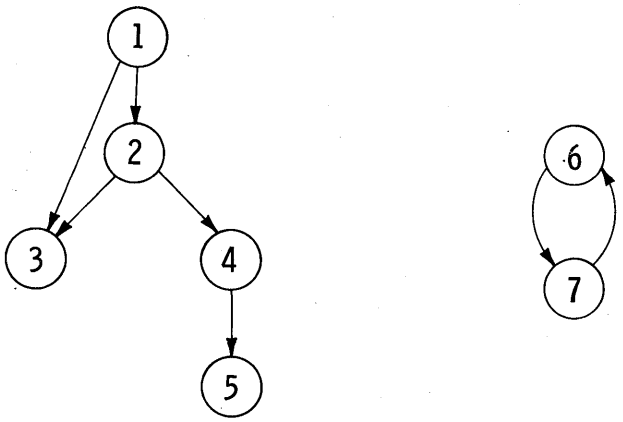


Figure 13—Net ordering relation

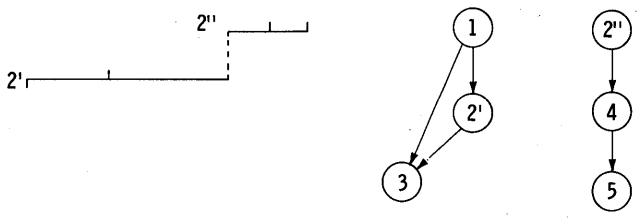


Figure 14—Chain breaking

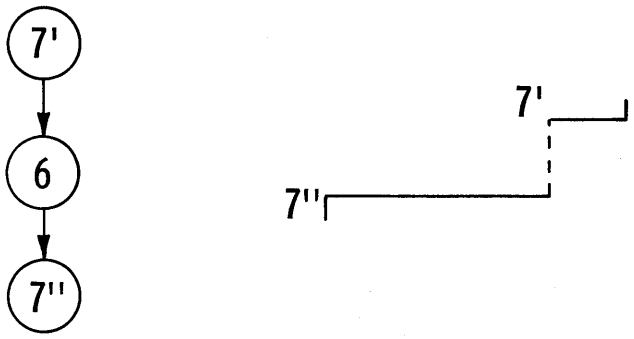


Figure 15—Cyclic conflict breaking

relation, (i.e., one segment is required to be wired above another) constrains the linepacking algorithm and, in some cases, leads to a set of constraints which are impossible to satisfy at all. Two such situations, the cyclic conflict and the chain, are illustrated in Figures 12 and 13.

It is clearly advantageous to remove these constraints, if possible, especially to cut all cycles and over-length chains. For this purpose, three straightforward methods are used. The first is to interchange logically equivalent pins of a logic block (e.g., two inputs of a NAND gate). The second is to introduce a short horizontal connection on the vertical plane which had the apparent result of moving the entry point of a segment into the street. The third is to wire a segment in more than one channel. By choosing an uncongested region of the street and dividing one horizontal segment into two segments with a vertical connector, it is possible to wire the two new segments in different channels and thereby satisfy previously impossible constraints. An example of such a solution is given in Figures 14 and 15.

The channel allocation program then scans the wire segments from left to right, filling one channel at a time. When one wire segment ends, the next nearest wire segment is selected to fill out the channel. This selection is conditioned by two considerations: the first is a conflict situation where a candidate segment is required to be above or below a segment which has not yet been wired. If the wiring of the candidate segment in the current channel precludes the wiring of the associated segment in the proper order, then the candidate segment is deferred. The second consideration is the affinity of the candidate segment to the side of the street in which the current channel lies. If two candidate segments are equally suitable from the point of view of linepacking, the one with the greater proportion of connections to that side will take precedence. Figure 16 illustrates the final results for the example shown in Figure 12. A "branch and

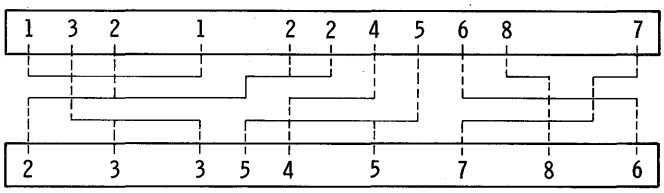


Figure 16—Line pack solution

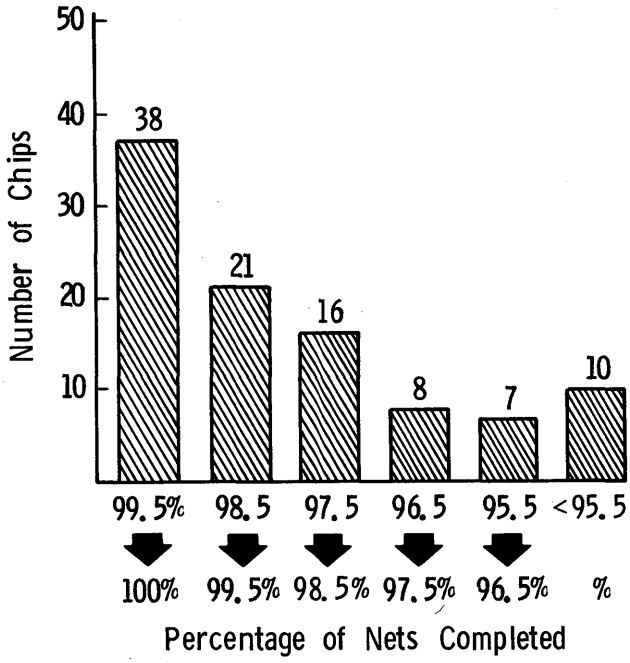


Figure 17—Histogram of completion percentages

bound" treatment of the bilateral channel allocation problem has appeared in the literature.[5]

RESULTS

A version of this method was implemented in 1972 to wire bipolar chips. The results quoted in Reference 6 will be summarized here.

A representative sample of 100 logic chips each containing an average of 125 logic gates and 136 nets. There were 10 vertical wiring avenues with eight channels each and 10 horizontal streets with four channels each. The programs ran in 325K bytes in an average of eight seconds CPU time on a 360/195. The wiring completion percentage in terms of segments, was better than 99 percent. The overall utilization of wiring space varied between 40 and 55 percent of the total length of metal which could be put on the chip.

The wiring programs exist within the framework of a larger physical design automation flow encompassing wirability analysis,[8] partitioning, placement,[9] interactive wire embedding[10] and checking. In almost half the cases, the design is fully automatic (Figure 17). In the rest, the interactive wire embedding program permits a designer to finish the wiring in several hours.

In the years since 1972, thousands of chips have been wired this way. These automatic wiring programs have been a key element in the success of the automatic design system of which they are a part and the chip technology which they support.

REFERENCES

1. Lee, C. Y., "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, September 1961, pp. 346-365.
2. Hightower, D., "A Solution Line Routing Problem on the Continuous Plane," *Proceedings of Design Automation Workshop*, 1969, pp. 1-24.
3. Hitchcock, R., "Cellular Wiring and the Cellular Modeling Technique," *Proceedings of Design Automation Workshop*, 1969, pp. 25-41.
4. Hashimoto, A. and J. Stevens "Wire Routing by Optimizing Channel Assignment within Large Apertures," *Proceedings of Design Automation Workshop*, 1969, pp. 115-169.
5. Kernighan, B., D. Schweikert and G. Persky, "An Optimum Channel Routing Algorithm for Polycell Layouts of Integrated Cirucits," *Proceedings of Design Automation Workshop*, 1973, pp. 50-59.
6. Chen, K. A., M. Feuer, K. H. Khokhani, N. Nan and S. Schmidt, "The Chip Layout Problem: An Automatic Wiring Procedure," *Proceedings of Design Automation Conference*, 1977, pp. 298-302.
7. Lee, J. H., "Aspects of Layout of Multi-Net in Multi-Layer Over Rectilinear Metric," *Ph.D. Thesis, University of Pittsburgh*, 1975.
8. Heller, W. R., W. F. Mikhail and W. E. Donath, "Prediction of Wiring Space Requirements for LSI," *Proceedings of the 14th Annual DA Conference*, 1977, pp. 32-42.
9. Khokhani, K. H. and A. M. Patel, "The Chip Layout Problem: A Placement Procedure for LSI," *Proceedings of the 14th Annual DA Conference*, 1977, pp. 291-297.
10. Habra, R. R., "Interactive Graphics for Wiring," to be published.

A speed-oriented, fully-automatic layout program for random logic VLSI devices

by A. FELLER and R. NOTO
Radio Corporation of America
Camden, New Jersey

INTRODUCTION

This paper describes a low cost, quick turnaround capability for generating high performance, random logic LSI and VLSI devices using the Standard Cell approach. This standard cell approach, described below, utilizes a fully automatic layout capability that automatically maximizes the speed of logic paths identified by the user as critical. In spite of the sophistication and size of the automatic layout program, the system can be run on Midicomputer based systems as well as time shared Main Frame Central systems. During the last 10 years over 1000 custom LSI devices have been fabricated using these techniques.

During this period the basic algorithms used in these automatic layout programs passed through a series of changes due, in no small part, to the needs and demands of a rapidly changing LSI technology. Three fundamental changes occurred during this period.

PLACEMENT, ROUTING, FOLDING PROGRAM

The first was the development of the Placement, Routing, Folding (PRF) program in the 1966-1968 time frame. An early PRF layout is shown in Figure 1. The basic approach used for placement and routing involved the generation of one dimensional geometry. In this algorithm all of the cells were placed in a linear row and optimally placed so as to minimize crossovers and total wire length. Then the array was literally folded by the computer program in a S-type function in order to achieve a rectangular aspect ratio. These algorithms were developed primarily for two phase dynamic PMOS logic.

SINGLE ENTRY TWO DIMENSIONAL PROGRAM (PR2D)

In the 1969-1971 time frame the second major change in the algorithms for the automatic layout programs occurred. This was the development of the single entry two-dimensional placement and routing program developed primarily for static logic. Its first application, and its principal one, was for generating random logic custom LSI devices using the metal gate CMOS bulk silicon technology. An illustration of a chip designed using this approach is shown in Figure 2.

With this program the placement function is based on a two-dimensional pair interchange of cells on the surface. Note, in Figure 2, that except for cells whose input terminals face each other, all connections between cells are routed around the ends of the cell rows in a wrap-around fashion. Because of this routing technique the distance calculation for optimum placement based on the two-dimensional pair interchange concept is based on an end around calculation as opposed to a direct orthogonal calculation.

As shown in Figure 2, significant chip area is devoted to the side channel wiring resulting from the wrap-around interconnections. It is clear that interconnections that passed directly through the cell rows, instead of around them, would result in a smaller, denser, and faster chip. This required far more sophisticated placement and routing algorithms and a more flexible design technique in the layout of the standard cell circuitry. Such an approach was developed in 1973 representing the third major change in the evolution of automatic layout programs for LSI and VLSI devices.

DOUBLE-ENTRY TWO DIMENSIONAL PROGRAM (MP2D)

This new approach is called the Multi-port Standard Cell Approach. This approach uses circuits called Double-Entry standard cells because unlike the previous approaches, connections may be made to these cells at the top or bottom of the cell. Figure 3 shows a silicon gate CMOS-SOS Divide by Two Counter Cell. This approach is particularly compatible with the silicon gate technology. Standard cell families have been developed for silicon gate on bulk silicon as well as sapphire substrates. The most significant advantage of this approach is that it now permits connections between cells anywhere on the chip to be made directly by permitting the connections to pass directly through the cell row. This is demonstrated in the silicon gate CMOS-SOS LSI device shown in Figure 4. Note the significant reduction in the side channel wiring with the resulting improvement in chip den-

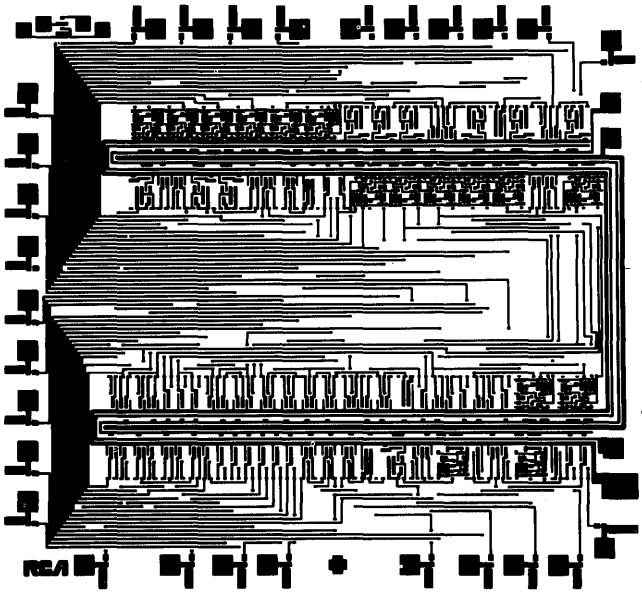


Figure 1—PMOS standard cell array—PRF program

sity and layout efficiency. Since 1973 this approach has been continually improved and is the basic technique that was used in the remainder of the LSI and VLSI devices that will be described or referred to in the remainder of this paper.

APPLICATIONS

Computer oriented applications for which these custom LSI capabilities are applicable include ROM decoders and sequencers, ALU's including special purpose ALU's such as an index register modifier, virtual memory control, dynamic memory management, polling and error detection and correction, general high speed random logic, various I/O functions particularly those associated with I/O computers etc.

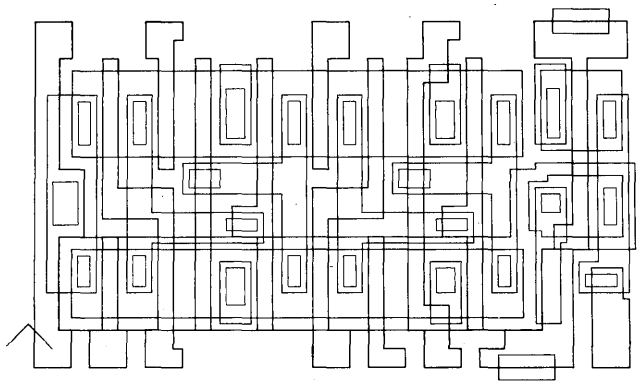


Figure 3—Double entry "divide by two" standard cell

SIMPLIFIED CAD SYSTEM

The block diagram of the CAD system for the Automatic Layout and Checking of LSI and VLSI devices which is shown in Figure 5 is a simplified version of the actual operating system although it does, for the most part, show the principal elements of the system. The only input required for the program is a net list containing some means of identifying the circuit or cell type used and the connectivity. Where there is a speed or propagation delay specification, that portion of the logic so identified will receive, automatically, special attention in the execution of the automatic placement algorithm so as to maximize the speed of those designated circuits.

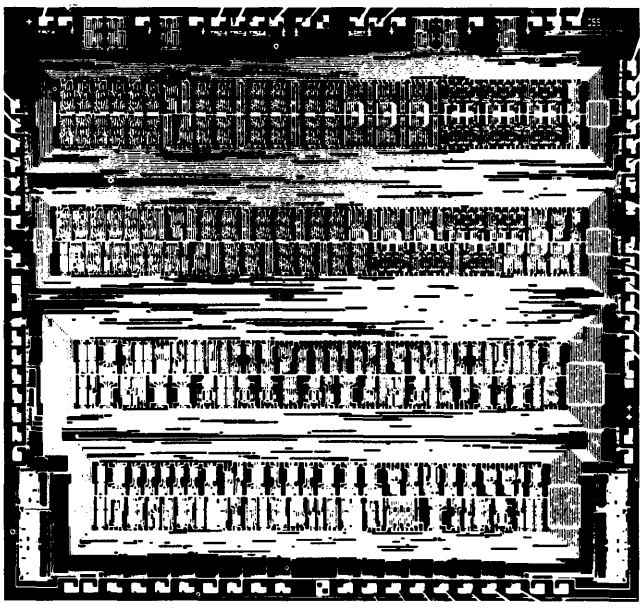


Figure 2—CMOS standard cell Chip—PRZD program

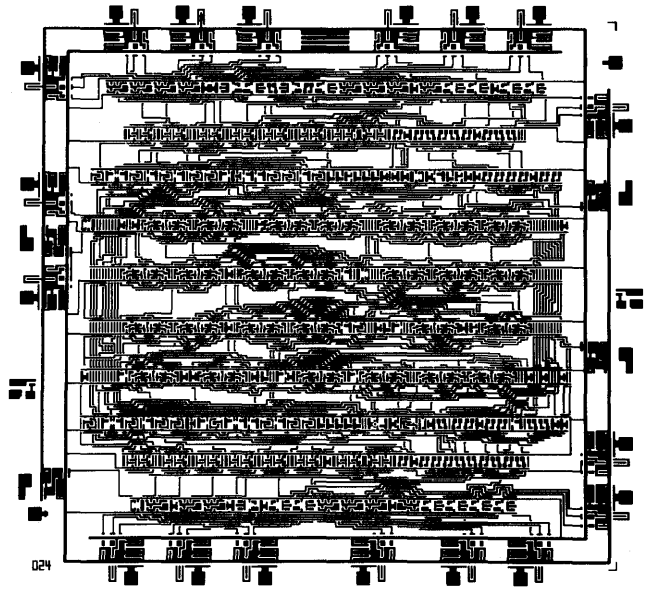


Figure 4—CMOS-SOS standard cell array—MPZD program

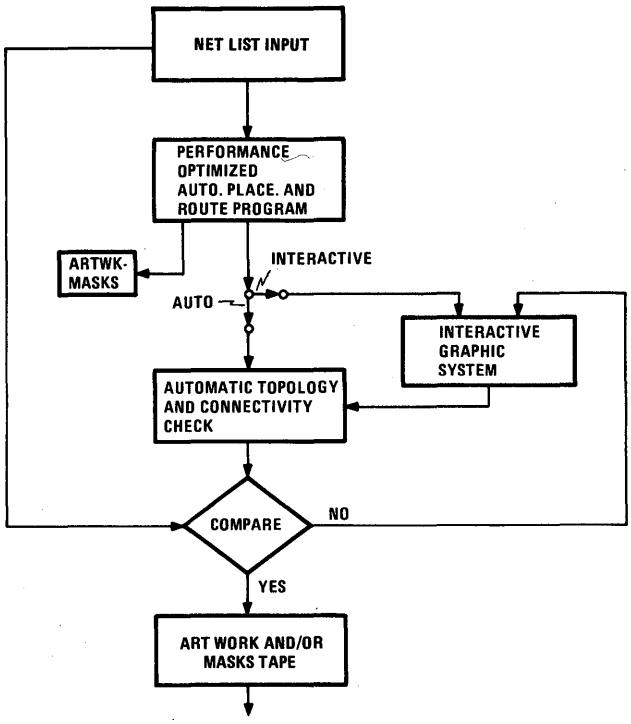


Figure 5—Automatic layout and check system

After the submission of the input data, the chip will be laid out by the Performance Optimized Automatic Layout Program. As shown in Figure 5 the user can exercise one of two options—the Automatic or the Interactive mode.

In the Automatic mode the output of the automatic layout program, as shown in Figure 5, can proceed to the artwork or mask fabrication phase either directly or through a series of topology and logic checking programs. In the interactive mode, the output of the automatic layout program is edited on an Interactive Graphic System (IGS). The designer may choose to edit the automatic layout for a variety of reasons ranging from area optimization to dynamic performance enhancement. The edited layout is then checked for any errors in either the topology or the logical connectivity. After validation, the system will automatically generate a magnetic tape to generate the final artwork and/or working masks.

STANDARD CELL LSI APPROACH

The standard cell LSI approach for generating LSI devices begins with the design, layout, and validation of a group of custom circuits called standard cells. The divide by 2 counter cell, whose topology is shown in Figure 3, is a typical example of a standard cell. Within the framework of the standard cell topology, all the inherent efficiency of a

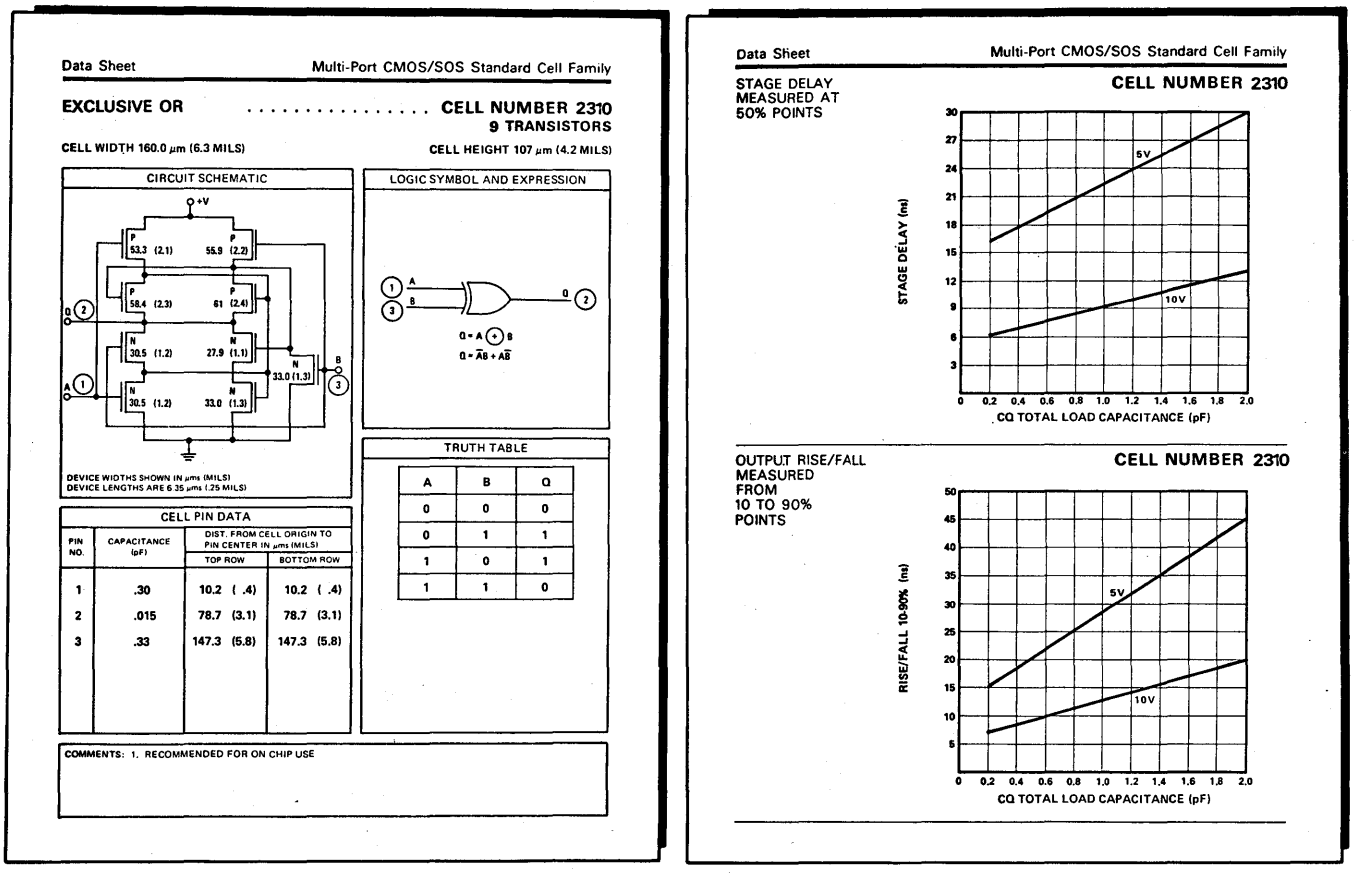


Figure 6—"Exclusive OR" standard cell data sheet

custom design may be incorporated into these individual circuit cells. Once validated, these cells are given an identification or pattern number and permanently stored. To use one of these cells in a logic design, the user calls for these cells by pattern number. To aid the logic designer in selecting the appropriate cells for optimum logic implementation, each cell is completely characterized and documented in standard cell data sheets such as that shown in the exclusive 'OR' function in Figure 6. The computer, under automatic program control, retrieves the cell data from the stored library in a way such that individual cells are represented by a collection of polygons corresponding to the various mask levels. The automatic layout program then places and interconnects the cells in accordance with the prescribed logic function.

DESIGNING LSI DEVICES WITH THE STANDARD CELL APPROACH

Designing LSI chips with the standard cell approach involves only the partitioning of the logic into standard cells, the identification of the selected cells and the generation of the net list or logical connectivity. This input data is expressed in a free-formatted, user oriented language called the Common Data Base specifically developed such that the input data can be programmed by a technician or design coordinator. Normally this is the only input data requirement. If, however, there are specific logical nets or paths whose speed or propagation delay are critical to the proper performance of the chip, such paths are identified in the input data. The placement algorithm will automatically ori-

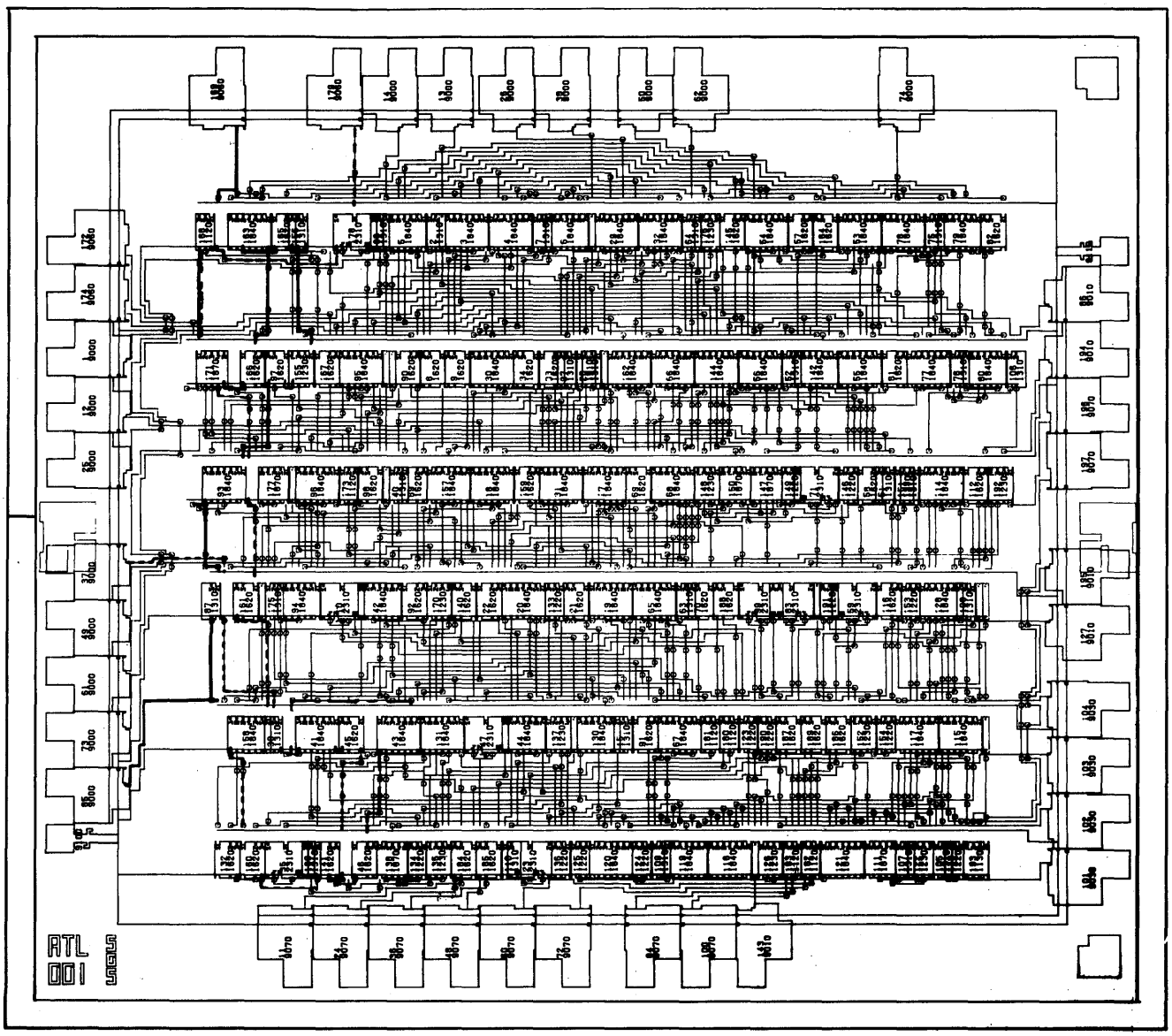


Figure 7—Critical path optimized automatic layout

ent the logic cells of the identified critical paths so as to minimize the delay associated with the interconnection parasitics. This is described in more detail in the following section.

PERFORMANCE AND SPEED OPTIMIZED CAPABILITY

As indicated earlier, a new capability of the LSI automatic layout program is its ability to minimize the propagation delays of logic paths identified by the logic designers as "critical" logic paths. Figure 7 illustrates this principle. This figure shows the computer generated layout of an 8-bit ALU with the critical paths highlighted. The closer spacing of the cells that form the critical paths results in reduced interconnection parasitics and hence shorter propagation delay. In addition to minimizing the delay of the selected 'critical' paths, the program also computes and generates a detailed listing of the parasitic resistances and capacitances associated with the interconnections. In the very high speed MOS technologies, such as CMOS-SOS, these parasitics are a determining speed factor. This information can be used to update the circuit simulation program or the 'race' and 'timing' computations of the Logic Simulation Program.

TABLE I—Improvement in Automatic Layout Program

Year	Size (mils)	Area ($mils^2$)	Density ($mils^2$/device)	Computer Cost	
				Minutes	Dollars
1974	146 x 121	17,666	18.2	Not Available	Not Available
1975	129 x 120	15,480	16.0	13.4	344
1976	115 x 111	12,765	13.2	5.4	132
1977	100 x 117	11,700	12.1	Not Available	Not Available

DENSITY AND EXECUTION TIME

In absolute terms substantial improvement has been made in increasing the density and reducing the execution time of automatically laid out standard cell LSI devices. This improvement in the speed and efficiency of the automatic placement and routing program is shown in Table I. Table I

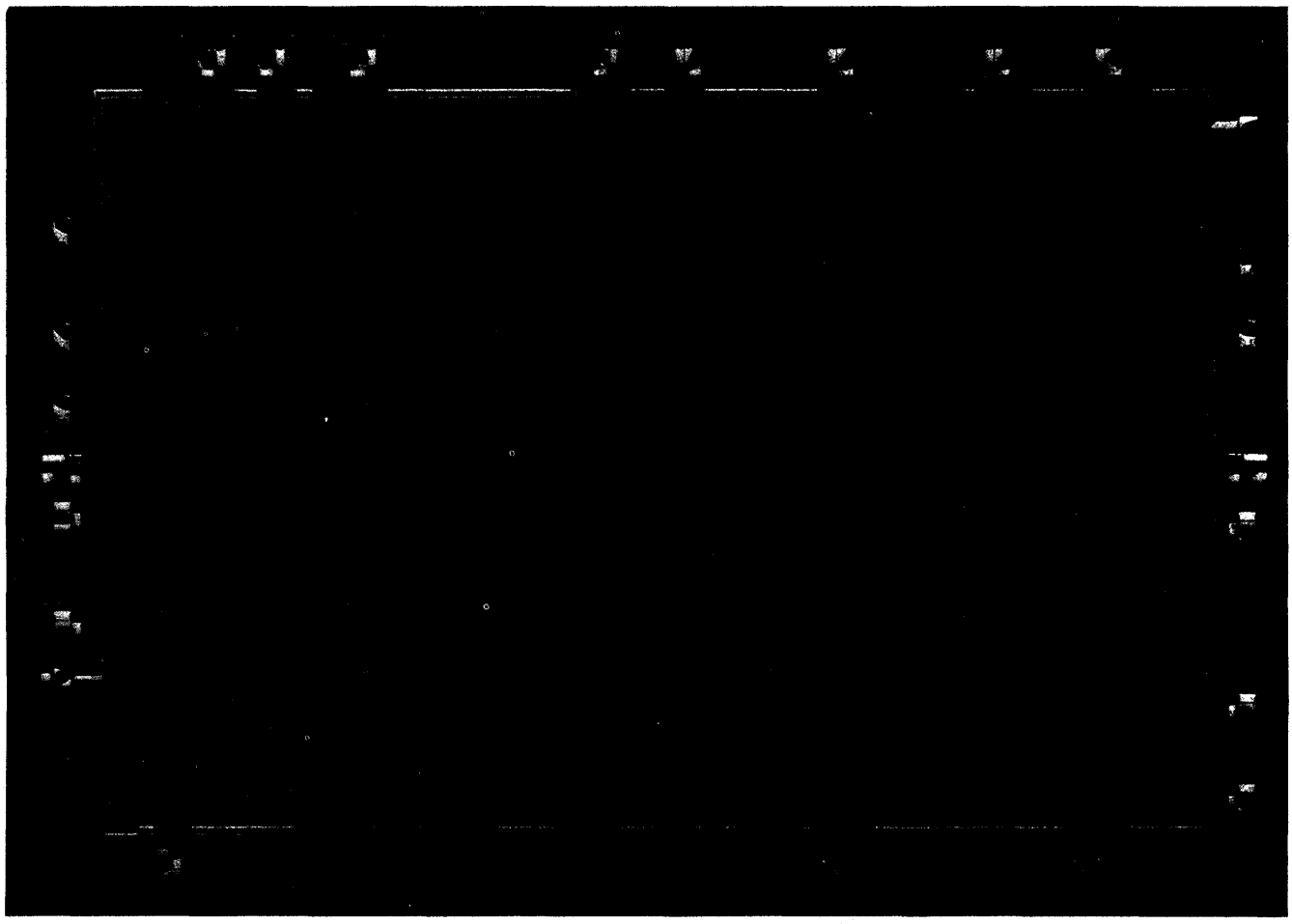
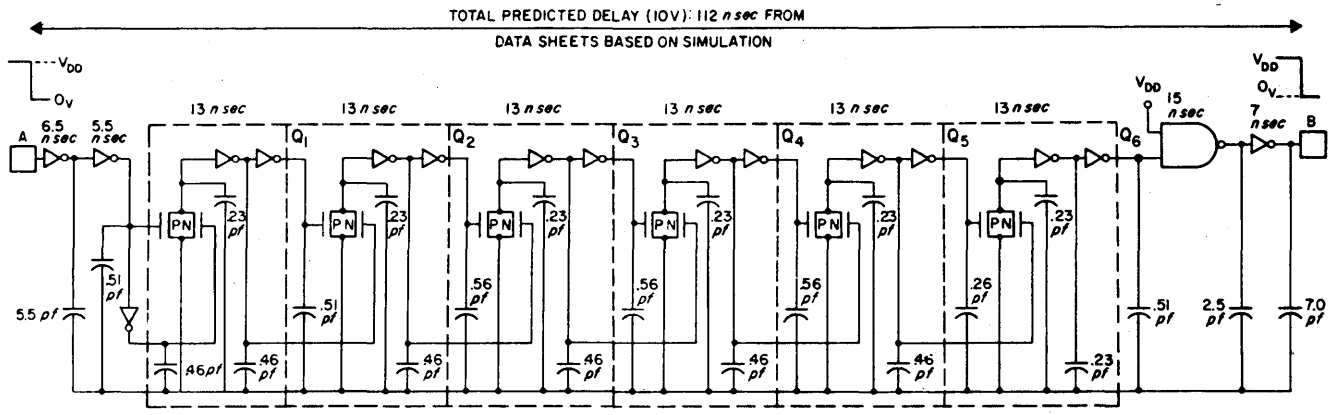


Figure 8



	AVERAGE GATE DELAY (nsec)				TOTAL INPUT TO OUTPUT (A to B)(nsec)				MAXIMUM COUNTER OPERATING FREQUENCY (MHz)			
	5V	10V	12V	15V	5V	10V	12V	15V	5V	10V	12V	15V
MEASURED	10	4.9	4.2	3.8	180	88	76	68	20	50		
PREDICTED	14	6.2	5.0	3.9	250	112	90	70			60*	80*

*EXTRAPOLATED

Figure 9—CAD generated LSI performance

shows the reduction in area (or increased density) and reduced execution time associated with the layout program since 1974. As shown in the table the area of the chip was reduced by 33 percent, while the execution time was reduced by 60 percent. These improvements are due entirely to improvements made in the efficiency of the placement and routing algorithms. The LSI device used as the test vehicle in Table I is a random logic device which is generally the type selected for implementation by the standard cell LSI automatic layout approach.

LSI AND VLSI APPLICATIONS

LSI application

Figure 8 illustrates a recent LSI application of the standard cell approach using the CMOS-SOS cell family. This chip, which was laid out with the automatic layout program, with virtually no manual editing, contains about 550 gates (2200 transistors) and has a chip size of 235×166 or about 17.65 square mils/device. Note that virtually all interconnections between cell rows pass directly through the cell rows. There are two basic methods which the layout program automatically uses in interconnecting cells on non-adjacent rows. If in a particular row there is a cell which is part of the same node or net as the source gate the program will use the polysilicon gate of the cell as a means of crossing the cell row. In this case, the polysilicon gate serves a dual purpose—that of a control element and a means of crossing a cell row. If there is no such cell on a particular cell row, the program will automatically move the cells in the appropriate area and provide a feedthrough cell to cross the intermediate cell row. As shown in the figure considerable area is associated with the large tristate drivers and protective devices in the pad area around the perimeter of the chip. The active area of the chip, essentially contained within the two parallel bus lines around the inside perimeter of the bonding pad area, measures about 202×132 square mils or about 12 square mils per device.

Although this chip is essentially a random logic device, it does contain sequential circuits such as ripple counters. Figure 9 shows the ripple path through a six stage counter in a form in which the path was analyzed by a circuit simulation program. The table in the lower half of Figure 9 gives not only the measured performance of this portion of the chip, but also the correlation between measured and computed performance. For example at 10 volt operating level the average predicted stage delay, based on conservative model parameter values, is 6.2ns while the measured average delay is 4.9ns.

VLSI applications (ATMAC microprocessor)

Two of the VLSI (>1000 gates) devices generated by the Standard Cell Automatic Placement and Routing program are shown in Figures 10 and 11. In addition to illustrating the VLSI capability of the standard cell approach, these chip types, which form the basic chip complement for a high

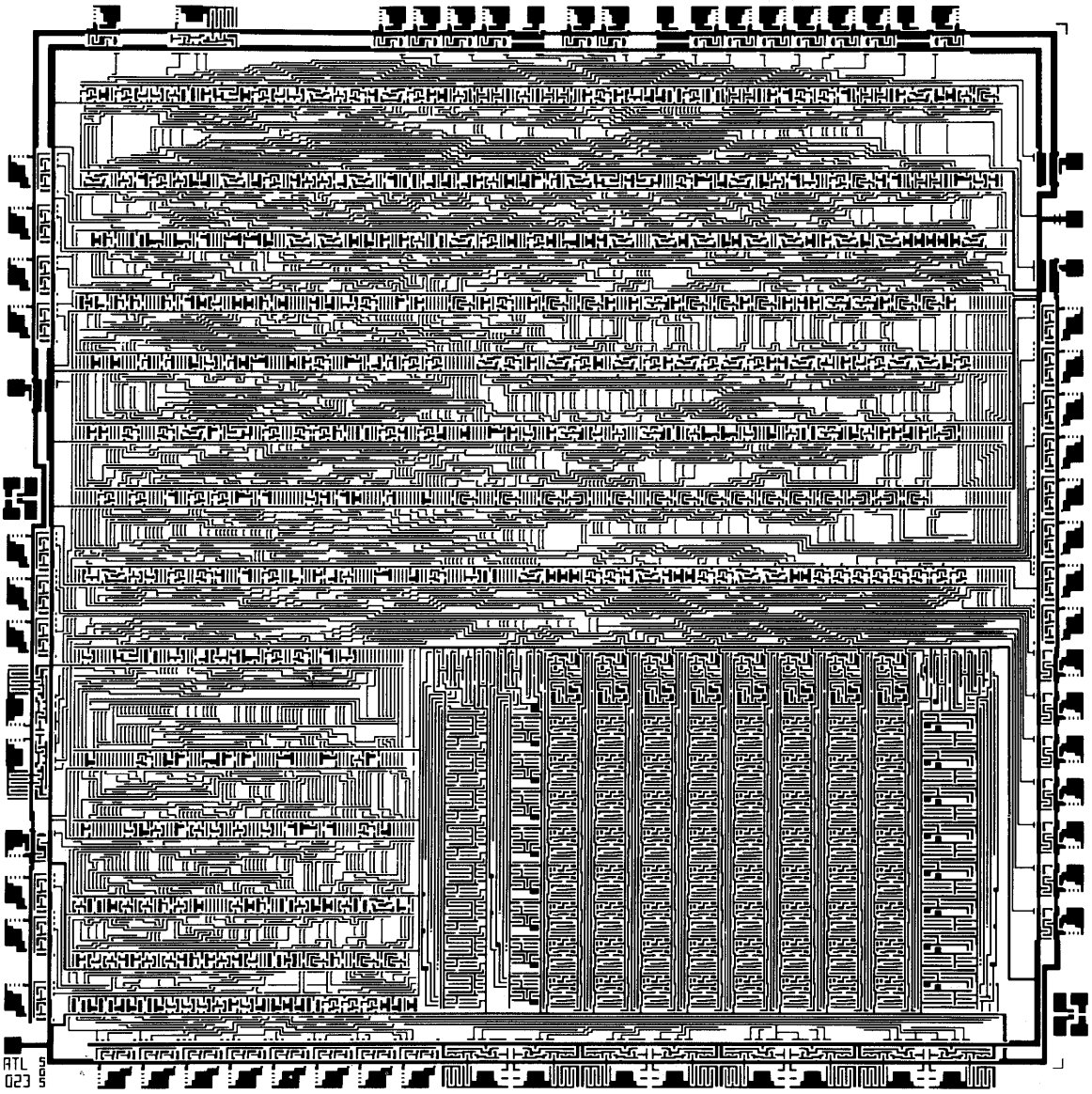


Figure 10—VLSI CMOS-SOS microprocessor chip #1

performance CMOS-SOS microprocessor, the ATMAC, were selected because they both illustrate a new capability of the Automatic Layout program called the Subchip Option. With the subchip option the program will accept a functional design, which may have been remotely designed by conventional manual techniques, (this includes designs on interactive graphic terminals) and automatically make all connections to it while simultaneously laying out the remainder of the host chip. The subchip option recognizes that certain logic configurations, such as memories, registers, counters, ROMs, and PLA's characteristically are regular and repeatable structures with a relatively simple interconnection matrix. As such it is not excessively costly and time-consuming to achieve a high density layout using conventional layout techniques as would be the case for complex, random logic devices.

The net effect of incorporating the subchip option into the standard cell automatic layout problem is to combine into this computer aided design capability the high density and optimization associated with handcrafted design devices with the low cost, quick turnaround capability associated with automatic layout programs such as the standard cell automatic placement and routing capability.

Each of the VLSI devices shown in Figures 10 and 11 contains about 5000 transistors, the equivalent of 1250 gates. Each chip contains both random and regular logic. The Data Execution chip, which performs the arithmetic processing and contains the data portion of the microprocessor, in-

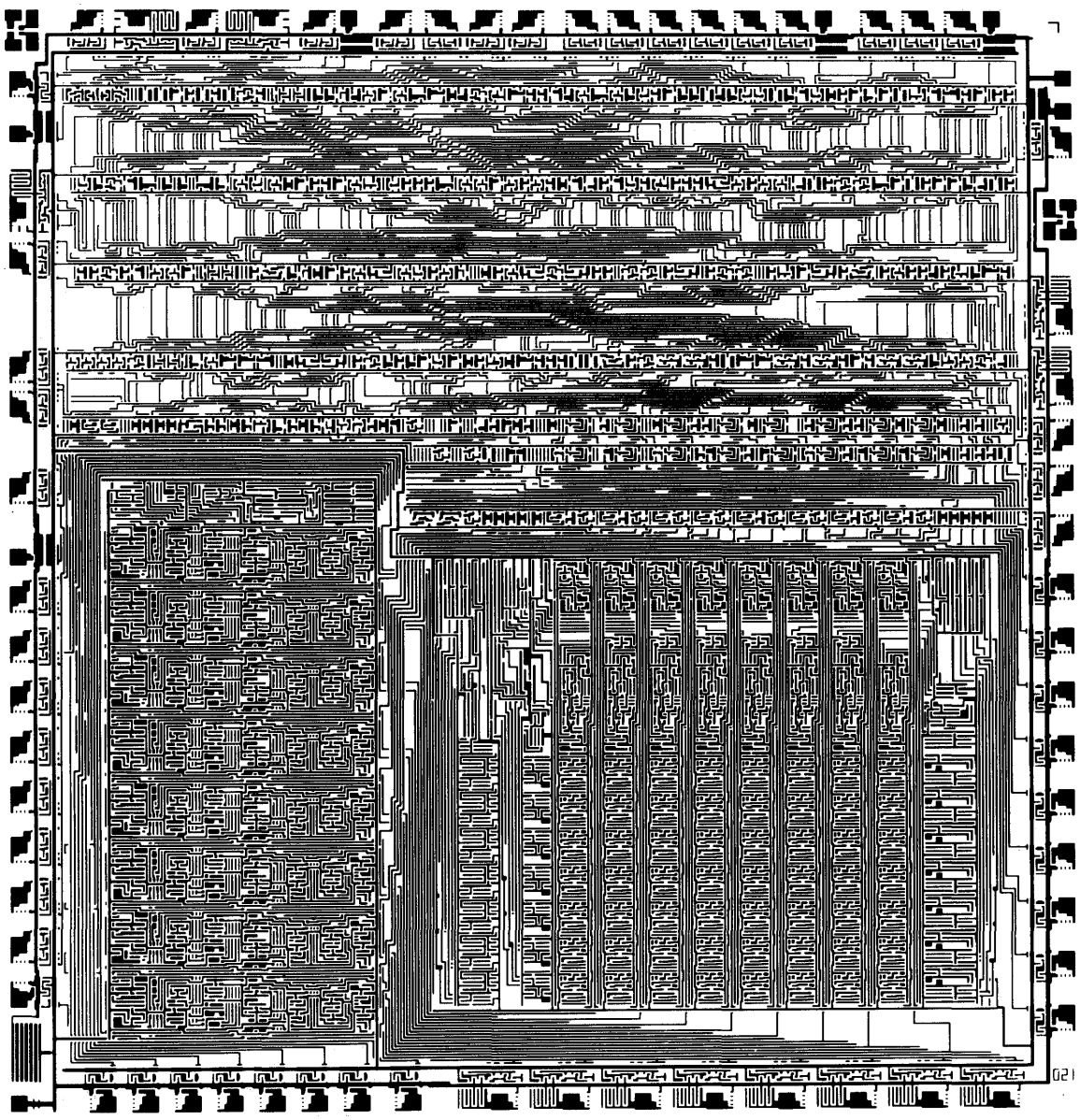


Figure 11—VLSI CMOS-SOS microprocessor chip #2

cludes one subchip function, an 8 word by 8 bit register stack. This function contains about 1200 transistors. The remainder of the chip, the random logic portion, which contains about 3800 transistors, was fully and completely laid out by the automatic layout program.

The other microprocessor chip, shown in Figure 11, is the Instruction and Operand Fetch (IOFU) chip, which functions parallel to the Data Execution chip. It is the control element of the system and operates control timing, program sequencing, operand accessing and array address updating.

As shown in Figure 11, this chip contains two subchip funcitons. Although one of these subchips is an 8×8 register stack, it is a different design from the one in the data execution chip. The other subchip is a 4 word by 8 bit Last In First Out (LIFO) register stack. Together these two handcrafted custom designs contain about 2000 transistors. The remainder of the IOFU chip, which contains about 3000 transistors, was automatically laid out and interconnected by the automatic layout program which automatically and simultaneously made all interconnections to the subchip functions as well.

The block diagram for this two chip 8 bit modular and expandible microprocessor is shown in Figure 12. It complements the CMOS/SOS technology with advanced parallel architecture to optimize its performance in signal processing applications. Its high speed/280 ns and 350 ns instruction times is enhanced by many architectural and programmable parallelisms that provide an effective throughput of greater

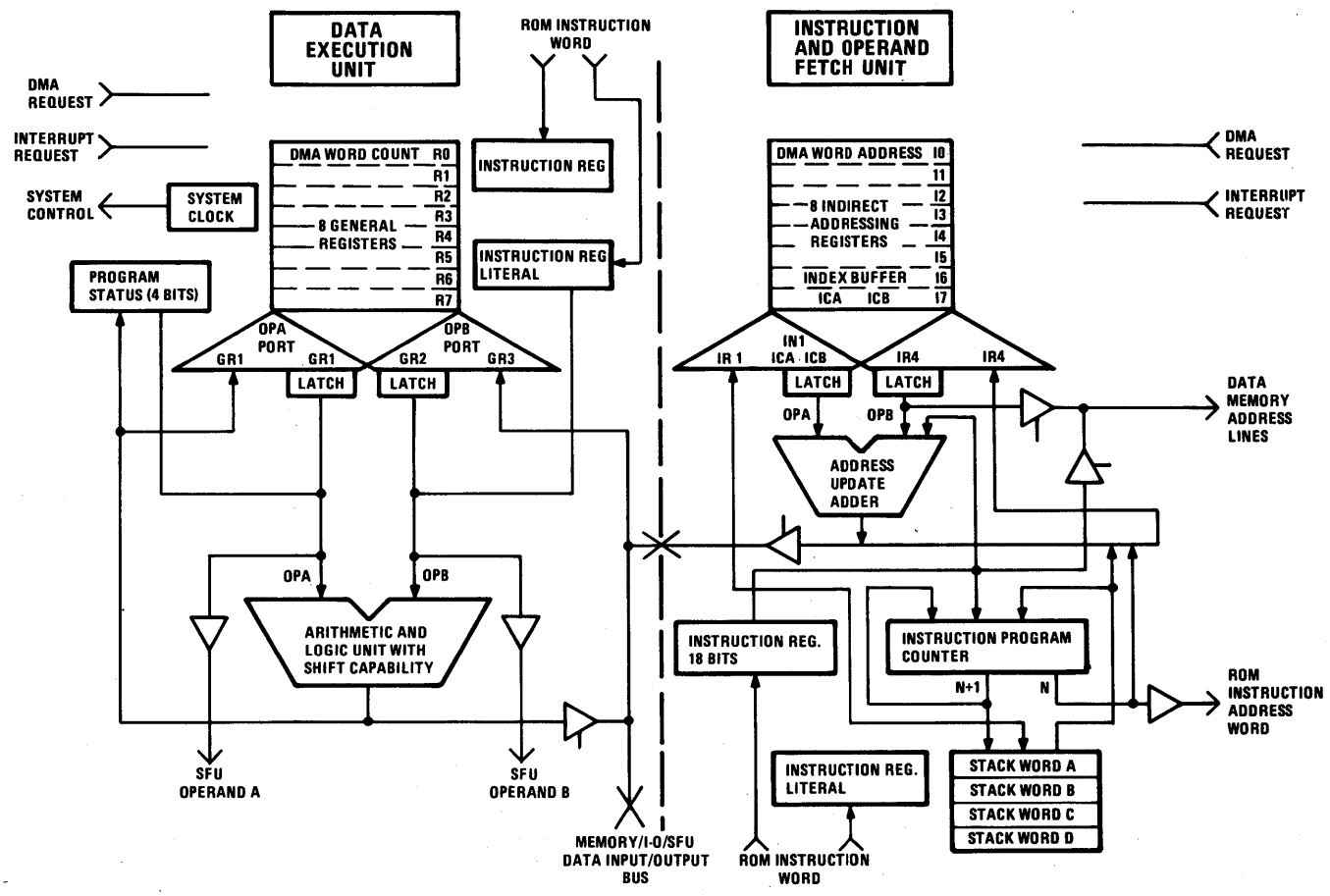


Figure 12—CAD generated CMOS-SOS microprocessor block diagram

than 10 million instructions per second in many array-type computations.

CONCLUSIONS

Because the standard cell approach for generating random logic custom LSI and VLSI devices is based on a completely automatic layout capability, it has proven to be a low cost, quick turnaround means of generating random logic custom LSI and VLSI devices, particularly for the larger and more complex devices.

With its low design cost and quick turnaround capability established, emphasis during the last few years has focused on improving its high speed performance. Two major improvements were made in the program affording higher density. One was improved placement algorithm that has resulted in a 30 to 40 percent improvement in density. The other was the introduction of the subchip option which combines into the standard cell approach the high density and custom optimization of handcrafted custom approaches with the low cost, quick turnaround capability that characterize the automatic layout capability.

ACKNOWLEDGMENT

A substantial part of the work reported in this paper was accomplished on programs supported by the Electronic Technology and Device Laboratory—ERADCOM—Fort Monmouth, N. J. under the direction of R. Reitmeyer. The authors gratefully acknowledge this support.

A methodology for design of digital systems—Supported by SARA at the age of one*

by GERALD ESTRIN
University of California at Los Angeles
Los Angeles, California

INTRODUCTION

There are many practical forces at work which encourage improvement in design methods. "Structured programming," "hierarchical design" and "modularity" are terms which have become commonplace. However the frequency of their use does not mean that design methods now exist which can guarantee that a particular complex system, whose programs are shared by many users, will have no design errors. Nor do design methods yet exist which can guarantee that a digital system, "simple" enough to be integrated into one device and reproduced in the millions, will have no design errors. It is well-known that almost all digital systems have so large a set of possible sequences of states that great ingenuity must be applied to achieve any meaningful degree of partial verification of their behavior. Theoretical advances continue to be made in verification. Those deriving from concepts of abstract data types[1] are incorporated in new languages[2-4] and have a lasting influence on the way we think about systems.

Despite those advances it is a source of great embarrassment to computer architects that semiconductor technology has created a capability which we cannot now exploit with proven design methods. A number of interesting approaches to design of concurrent systems are being actively pursued. However we do not currently have the languages, operating systems and design methods needed to effectively employ the new LSI devices which can now be produced. We would like to be limited only by economic factors, not technical or theoretical factors.

One hope is that there will evolve effective methods for composing such systems from defined, well behaved building blocks whose composite behavior can be shown to meet prestated requirements. A second hope is that starting from well formulated requirements and an initial abstract solution system, there will evolve helpful guidelines for structural partition and effective algorithms for behavioral refinement. The refinement procedure should conserve desirable properties through as many levels of abstraction as the design needs. A third hope is that a new dimension for architecture of computer systems will emerge from these design methods and permit us to effectively use the outpouring of large scale integrated devices.

Realization of these hopes and their joint use are the goal of the methodology discussed in the first part of this paper. The second part of the paper describes the state of the system implemented to support a designer attempting to use the methodology. A companion paper[5] illustrates application of our methods to design of software.

The System ARchitects Apprentice, SARA, is only one year old. The roots of this work were reviewed in short by the author in one of a set of presentatons[6] which announced SARA early in 1977.

We are not so simplistic as to think that we have THE only right way to design computer systems. We do feel that our methodology makes some significant advance in approach to design of concurrent software and hardware systems and has great promise for incorporation of previously verified models of software or hardware systems. In fact it has long been recognized that methods for design of concurrent systems are also important to good sequential system design. The models used to represent desired concurrency can also be used to model a set of processes which are mutually independent. Sequence can then be imposed on them after appropriate analysis rather than by chance.

As verification methods become more effective and succeed in avoiding combinational explosion[7] we hope to combine them with the UCLA design methodology and move closer to effective use of semiconductor device capability.

The UCLA design methodology

We tersely characterize the UCLA design methodology as being requirement-driven and supportive of self-documenting design of modular, concurrent, hardware and software systems. Figure 1 displays a high level flow chart of the design procedure. We will proceed through each step discussing what is done with particular attention to implied analysis or value judgments made outside of the stated procedure. In this first part of the paper the reader need make no assumption about the nature of automation aids but rather should focus on the systematics.

* This research was supported by the U.S. Department of Energy, Contract No. EY-76-S-03-0034, PA 214.

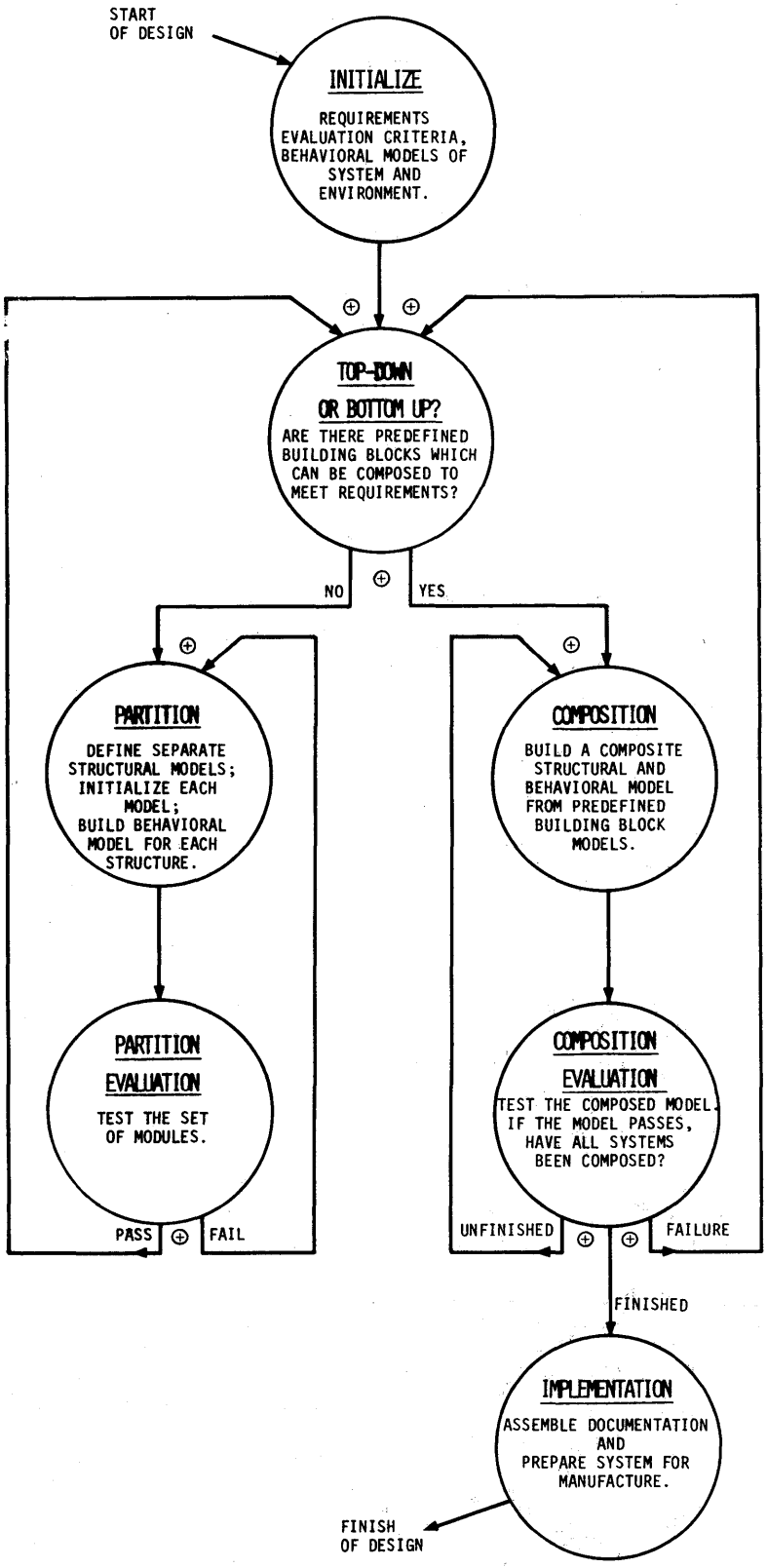


Figure 1—UCLA design methodology

Initialization of a design

This design methodology is characterized as requirement driven. Given a set of requirements and assumptions about the environment, the designer conceives of a system to satisfy those requirements. The bulk of our discussion is about a procedure which develops sufficient detail about the system behavior to determine which of the requirements are not satisfied.

Requirements are not God-given and there continues to be extensive study of disciplined analysis techniques[8,9] leading to well formed requirements. When design costs get too high it is often necessary to reconsider the requirements themselves. For purposes of this paper we will assume that requirements have been carefully generated and that we are to make our best effort to design a system which satisfies them. It is generally recommended that requirements be made at a sufficiently abstract level that they do not prematurely commit to particular implementation methods. However, many designers make certain implementation decisions early for practical reasons. For example, a manufacturer may restrict implementation to a small prescribed set of well tested, well understood building blocks, in order to minimize inventory and training costs. These must then be carried through the rest of the design process and not be compromised.

Requirements fall into different classes with respect to their use in this design procedure.

Some requirements carry quantitative design constraints, such as real time response to a stimulus, which lead directly to use of analytical methods or to generation of experimental tests. These are the most straightforward requirements to deal with, with the exception of two problems:

- A numerically stated requirement often implies more precision than was intended and it is sometimes tedious and difficult to introduce tolerances or to manage tolerances during evaluation.
- When a high level system design is partitioned it is necessary to determine the set of subsystems which are involved in satisfying a quantitative requirement and, where possible, to estimate a new quantitative subrequirement for each subsystem. Such estimates can turn out to be quite arbitrary and to produce undesirable distortions of the design. This is a poorly understood art.

Many requirements are qualitative. In some cases their satisfaction is determined by delegating the responsibility for judgment to individuals who can then record approval or disapproval. In other cases qualitative requirements are simply passed on to the fabrication phase because they cannot be evaluated until manufacture. Qualitative requirements deserve careful attention because they are often the focus of user dissatisfaction. When tradeoff analysis can be made to show optimality of a design response to a qualitative requirement it can save a great deal of misunderstanding and expense.

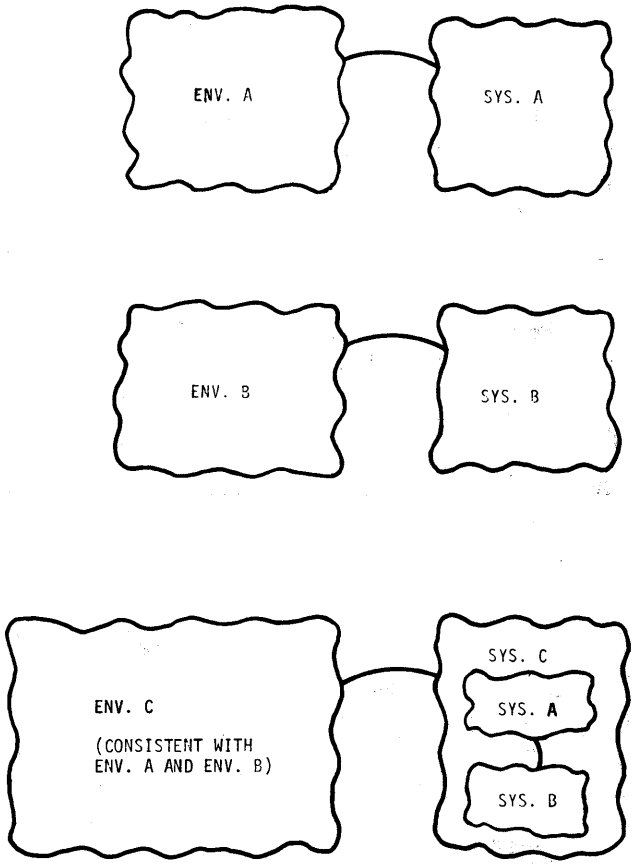


Figure 2—Composition of systems and environments

An integral part of our design methodology requires initial behavioral modeling of the system under design and its assumed environment. The environment model displays only those properties necessary to constrain the system design and is constructed using the same primitives as those used for models of systems being designed. Therefore when two independent designs are composed, a designer must test the consistency between each subsystem and the others' environment, as informally shown in Figure 2.

Partition or composition decision

The decision to compose a model of the system directly out of predefined models or to go through another stage of top down partition depends upon the availability of predefined building blocks and the designers understanding of their behavior. A designer may have a number of building block models, which seem to fit a particular design jigsaw, but be missing one type of building block. Rather than pass through a uniform top-down partition, which always entails some overhead, the designer may choose to move aside into

an independent design phase. There the designer would establish requirements for the new building block, successfully complete its design and validation using existing building blocks, and then return to complete composition along the original design path.

It is of value to add to our intuitive acceptance of what building blocks are by the following definitions:

By building blocks we mean physically realized systems with:

—constrained environments, and
—predictable input-output behavior,

such that, given:

—two or more such compatible systems, and
—connectivity constraints between them,

then the combined input-output behavior is predictable. Whenever a combined system is also a building block we call it well behaved.

Building block models are representations of building blocks if:

—each model has one or more defined physical realizations satisfying the definition of a building block, and
—when building block models are interconnected consistent with interconnection constraints, then combined corresponding physical realizations form a physical realization of the combined model.

Composition

In the composition step the full set of selected building block models in interconnected—to each other and to the assumed environment model. Attributes, associated with the inputs and outputs of each building block model, provide the basis for consistency and completeness checks. These checks must include an environment check as noted above for Figure 2. The complete model is then analyzed or experiments are conducted in a simulation environment.

If implementation decisions have not been made earlier they are introduced in the composition step building block models.

Composition evaluation

The evaluation step consists of a combination of analysis and experiment. If the behavior of each building block has been verified, it may be possible to verify a system requirement which has been formally stated. For example, if one of the requirements states that a concurrent system must satisfy "proper termination" then it is possible that analytical tools[10,11] can be used. More generally the designer will generate experiments in a simulation environment and choose the most stressful conditions for test.

Partition

A partition procedure is used when a set of requirements cannot be met directly by composing known building blocks. There are different strategies for guiding partition of a system into subsystem modules.

- Parnas[12] has noted that modular decomposition may be done according to independent user functions which need to be implemented or according to system functions which are needed in carrying out every user function. The former approach may make mapping of requirements more direct. The latter approach generally leads to better utilization of shared resources.
- One module may be separated from another because both design and observation of the included operations are considered vital to performance, reliability or cost. The module is a system element which is most easily observed because its interface with the outside is made explicit.
- A set of modules may be selected because it is estimated that they all have about the same complexity, thereby providing a design balance.
- A set of modules may be selected because it is estimated that they all have approximately equal influence on performance and therefore represent an educated guess of a balanced system.
- Modules may be selected because they are readily composed from known building blocks and therefore will avoid another partition step.
- Modules may be selected because they simplify refinement of higher level behavioral models in a manner which guarantees preservation of desired properties. Thus, consistency with invariants established in higher level proofs of data flow or control flow properties would be example criteria.

Whichever combination of guidelines are practiced leads to the creation of two or more structures and their interconnection. For each named subsystem there is then an initialization step identical to that carried out at the previous level with the exception that the higher level requirements must be mapped onto the new structures. For each named subsystem a behavioral model is developed which seeks to meet the subsystem requirements and hopefully to join in meeting full system requirements.

Partition evaluation

The set of subsystem models are evaluated by inspection, by analysis or by simulation. If any evaluation fails, the design is returned to the partition phase for modification or re-creation by the designer. If all evaluation steps pass then the design is returned to the part of the process where, for each subsystem, it is decided that composition can take place directly or else that another partition is needed.

Implementation

When the compositions succeed in passing all tests for all subsystems, the design moves into an implementation phase to prepare the design for fabrication. The following operations are generally needed:

- Documentation is prepared. If there have been careful records kept during the design process, they provide the best description of intended functions, predicted behavior, and the usually critical interfaces between subsystems. This documentation is important for those who will evaluate the fabricated system. However other kinds of documentation are generally needed to facilitate fabrication, maintenance, marketing, training and distribution. Wiring lists, microprograms, bit maps for read-only memories, higher level language programs and test programs are examples.
- Changes may have to be made to remove artifact introduced by the modeling and design process. For example, instrumentation may have been integrated with a model to determine whether performance requirements were met but the measurements may themselves deteriorate performance and not be deemed necessary. This excision must be done very carefully in order not to alter desired behavior and sometimes it will force another pass through evaluation of the altered model.
- The simulation environment may have mechanisms which cannot be directly fabricated and are replaced by "equivalent" mechanisms. This too must be done very carefully lest the modeling and evaluation process be invalidated.

THE SARA (SYSTEM ARCHITECTS APPRENTICE) SYSTEM

The groundwork for SARA was set by Gardner*[13] at the beginning of 1975 following several years of collaboration between Gardner, Potash and Estrin. The design of SARA was begun during the summer of 1975. Implementation of structural and behavioral modeling tools continues through the present. SARA's birth was announced at the beginning of 1977[6] and a one year old version now marks SARA-reborn at MIT-Multics, a more permissive environment than UCLA's IBM/TSO. The infant version of SARA provides a collection of modeling tools, at both UCLA and MIT, for which some care has been applied to the design of the user interface. The sibling SARA's are reachable by authorized users through the ARPANET.

The current view of the structure of the SARA system is shown in Figures 3 and 4.

* In addition to R. I. Gardner, major contributions to the design and implementation of SARA were made by D. Berry, I. Campos, C. Carper, J. Drobman, B. Fenchel, W. Overman, R. Razouk and W. Ruggiero.

User interface

All communication between a designer and the SARA DESIGN SYSTEM as well as output to FABRICATION are done through IO.[14] The IO system creates a flexible and powerful user-interface and achieves uniformity of interface with the various tools in SARA. Moreover it serves as a place to contain all machine dependent routines, easing any move from one PL/1 environment to another. SELECTOR[14] is then used to move between language processors at different levels as illustrated in Figure 5.

Structural and behavioral modeling tools

The most fully tested tools are those contained in the modules named STRUCTURES and BEHAVIORS.

- STRUCTURES holds a language processor called SL1 which lets a designer interactively describe the structure of a system using three primitives (see Table I): modules, sockets and interconnections. Multilevel structural models can be constructed using fully nested modules as in Figure 6. Multilevel interconnections, can be expressed simply, as displayed in Figure 7. In general interconnections may be refined as complete systems.
- BEHAVIORS holds several language processors used to create different kinds of models of named structures. The principal behavioral modeling tool is the Graph Model of Behavior (GMB). The GMB provides a small number of primitives (see Table II). The control graph primitives allow a designer to interactively create a graph explicitly modeling flow-of-control among processes including synchronization of concurrent processes. The data graph primitives allow a designer to display flow-of-data, interpreting the processes in the control graph at an abstract level. Furthermore a preprocessor, called PLIP[15,16] allows a designer to concretely interpret each data processor and dataset primitive using the full power of PL1. The topology of both the control and data graphs is fixed and they are used to model pseudo static aspects of system behavior. The PL1 interpretation (PLIP) models dynamic behavior within each processor and causes changes in control and data flow along the prescribed data and control arcs. Two other tools have been built for analysis of the GMB model. An interactive simulation environment[16] permits experiments to be executed, observed and analyzed. The simulation environment is used to check predictions of results and estimate timing. Moreover, the GMB simulator is effective at detecting deadlocks, detecting contention for datasets and detecting contention for processors. One reduction program is used to analyze the control flow graph and this analysis can determine that a system is "properly terminating", i.e., that it is deadlock free and reentrant.

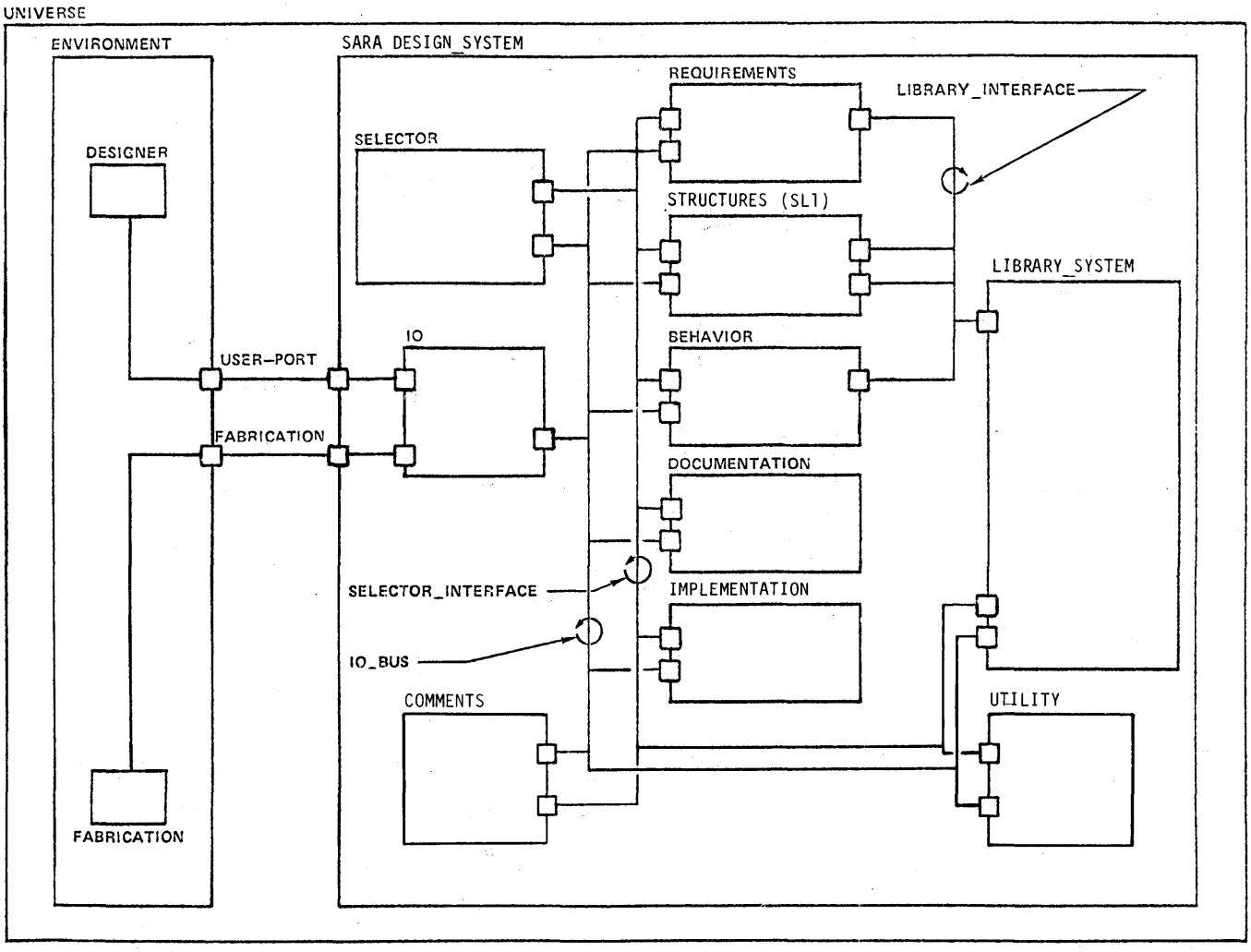
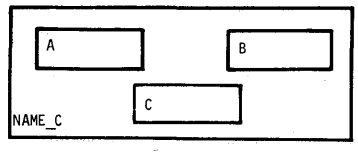
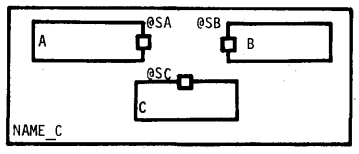
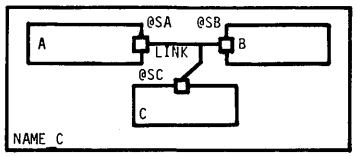


Figure 3—Structure of SARA

TABLE I.—Modeling Primitives

TYPE	GRAPHICAL	MACHINE PROCESSABLE
STRUCTURAL PRIMITIVES		NOTE: SL1 STATEMENTS MAY BE MADE IN ANY ORDER. THEY CAUSE CREATION OF IMPLIED STRUCTURES.
A NAMED MODULE REPRESENTS AN OBJECT WHOSE INTERNAL, FULLY-NESTED STRUCTURE IS HIDDEN FROM THE OUTSIDE. A MODULE'S ONLY POSSIBLE COMMUNICATION WITH THE OUTSIDE IS THROUGH A SOCKET. OTHERWISE, A MODULE'S NAME IS KNOWN ONLY TO THE STRUCTURE WITHIN WHICH IT IS NESTED. EXAMPLE: THE MODULE, CALLED "NAME_C", IS COMPOSED OF MODULES "A" AND "B".		NAME_C (A,B,C)
A NAMED SOCKET IS PART OF A MODULE BUT THE SOCKET'S NAME IS KNOWN BOTH INSIDE AND OUTSIDE OF ITS HOST MODULE. EXAMPLE: THE MODULE "NAME_C" IS COMPOSED OF MODULES A WITH SOCKET SA, B WITH SOCKET SB AND C WITH SOCKET SC.		NAME_C (A@SA, B@SB, C@SC)
A NAMED INTERCONNECTION IS A STRUCTURE WHICH CONNECTS TWO OR MORE SOCKETS. EXAMPLE: AN INTERCONNECTION CALLED "LINK" CONNECTS THE SOCKETS A@SA, B@SB, AND C@SC.		LINK + (A@SA, B@SB, C@SC).

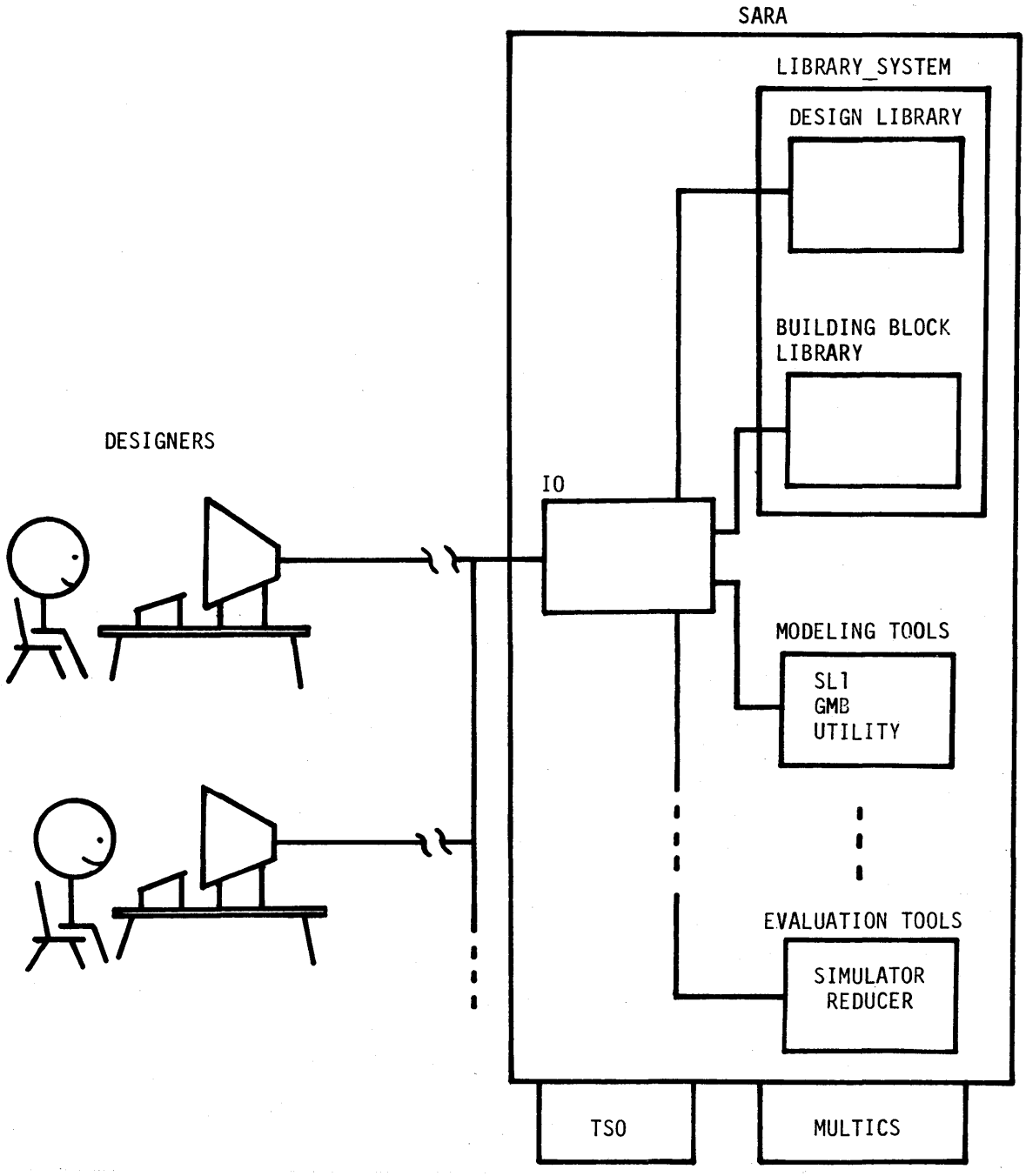


Figure 4—Designers view of SARA

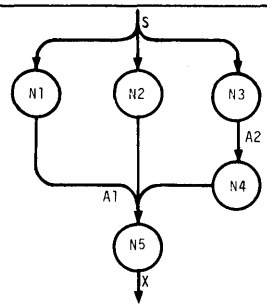
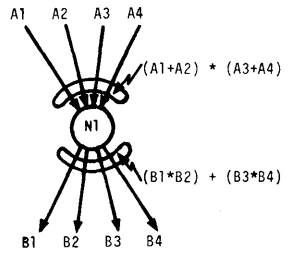
Three other behavioral modeling tools are in various stages of integration in SARA.

A simulation system called DCDS (Digital Control Design System) was implemented in 1969[17] and improved in the early 1970's. DCDS is not interactive but allows a designer to richly describe a system in terms of logic nets, in terms of microprogram sequences or in terms of algorithms. A fourth sublanguage called DECLARE is used to combine models described separately in the three languages: LOGIC, MICROPROGRAM and PL1. DCDS is not integrated into the set of interactive SARA tools. Models are translated in a batch mode and simulations run in a batch mode.

The Instruction Set Processor (ISP) modeling language created by Bell and Newell[18] has attracted widespread interest. Buliding upon work by Crocker[19] a translator from ISP to PLIP was created. One complex machine model was built and is still undergoing test.

The Simulation Oriented Language (SOL) of Knuth and

TABLE II.—Behavioral Modeling Primitives

TYPE BEHAVIORAL PRIMITIVES	GRAPHICAL	MACHINE PROCESSABLE
A NAMED CONTROL NODE REPRESENTS A STEP IN A PROCESS BEING MODELED. A CONTROLLED DATA PROCESSOR (SEE BELOW) MAY BE ASSOCIATED WITH A NODE TO PROVIDE INTERPRETATION OF THE PROCESS. EXAMPLE: A NODE N1 HAS A SINGLE ENTRY ARC S AND A SINGLE EXIT ARC X.		@CONTROL_GRAPH; @NODES N1; @ARCS S,X; N1 (S:X); @END;
A NAMED DIRECTED CONTROL ARC REPRESENTS NON-VOLATILE PREDEDENCE RELATIONS BETWEEN SETS OF NODES. IF THERE IS MORE THAN ONE SOURCE NODE OR DESTINATION NODE THE ARC IS CALLED COMPLEX; OTHERWISE IT IS CALLED SIMPLE. AN ENABLING TOKEN IS PLACED ON AN ARC EITHER AS A STARTING STATE OR UPON TERMINATION OF ANY OF ITS SOURCE NODES. WHEN A NODE IS INITIATED, ITS ENABLING TOKENS ARE ABSORBED. EXAMPLE: A2 AND X ARE SIMPLE CONTROL ARCS. A1 IS A COMPLEX CONTROL ARC WHOSE SOURCE SET IS NODES N1, N2 AND N4 AND WHOSE DESTINATION SET IS N5, S, IS AN INCOMING COMPLEX ARC WHOSE DESTINATION SET IS N1, N2 AND N3.		@CONTROL_GRAPH; @NODES N1,N2,N3,N4,N5; @ARCS S,A1,A2,X; N1 (S:A1); N2 (S:A1); N3 (S:A2); N4 (A2:A1); N5 (A1:X); @END;
INPUT CONTROL LOGIC A LOGICAL RELATION AMONG THE INPUT ARCS TO A NODE SPECIFIES THE PRECEDENCE CONDITIONS THAT MUST BE SATISFIED BY TOKEN STATES FOR THE NODE TO BE INITIATED. TOKENS FROM THE INITIATING ARCS WHICH SATISFY THE INPUT RELATIONS ARE ABSORBED BY THE TOKEN MACHINE. TOKENS ARE ABSORBED FROM ONE OF AN INITIATING ARC SET GOVERNED BY AN OR RELATION IN A MANNER ESTABLISHED IN THE TOKEN MACHINE AND FROM ALL MEMBERS OF AN INITIATING ARC SET GOVERNED BY AN AND RELATION. EXAMPLE: IF ENABLING TOKENS EXIST ON EITHER A1 OR A2 AND ON EITHER A3 OR A4 THEN N1 CAN BE INITIATED. OUTPUT CONTROL LOGIC A LOGICAL RELATION AMONG THE OUTPUT ARCS SPECIFIES WHICH ARCS HAVE TOKENS PLACED UPON THEM WHEN A CONTROL NODE IS TERMINATED. WHEN AN EXCLUSIVE OR OUTPUT RELATION HOLDS, A DATA PROCESSOR INTERPRETATION MUST DECIDE WHICH ARC RECEIVES A TOKEN. WHEN AN AND RELATION HOLDS ALL OUTPUT ARCS RECEIVE TOKENS. EXAMPLE: WHEN N1 TERMINATES, ITS ASSOCIATED CONTROLLED DATA PROCESSOR WILL HAVE DECIDED WHETHER TOKENS ARE TO BE PLACED ON B1 AND B2 OR ON B3 AND B4.		INPUT:OUTPUT CONTROL LOGIC @CONTROL GRAPHS; @NODES N1; @ARCS A1,A2,A3,A4,B1,B2,B3,B4; N1((A1+A2) * (A3+A4): (B1+B2) * (B3+B4)); @END;

McNeely[20] was implemented in PL1 by the Defense Communication Engineering Center under the name SOL/370. It has been installed but not fully tested and integrated. It is proposed for use as one of the SARA behavioral model analysis tools.[21]

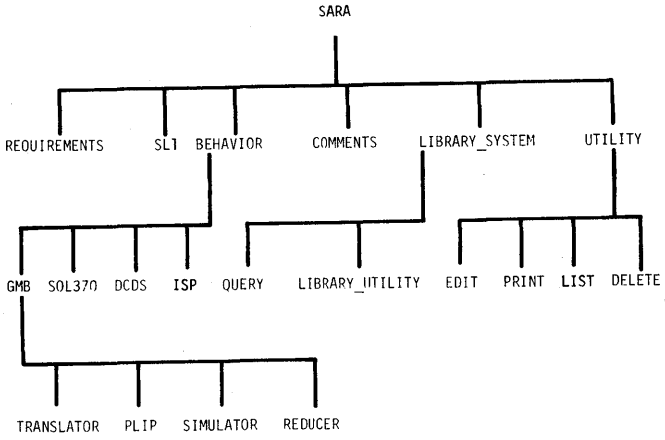


Figure 5—SARA—Multilevel view from the selector

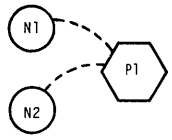
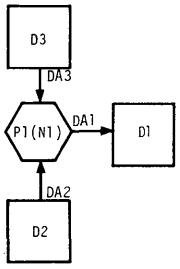
The design library

Every tool in the SARA system makes use of LIBRARY SYSTEM for storing and retrieving models. There are two essentially different parts of the LIBRARY SYSTEM.

- One part contains the public building block library which holds models of hardware and software subsystems that have been extensively tested or verified. These models have a great deal of value added and are assumed to be called upon strongly enough to make the investment pay off. Extensive controls on entry and modification will be exercised by a library administrator. Drobman has considered the nature of the library system with respect to hardware system design and has proposed generic building block structures for LSI based systems.
- A second part provides facilities for storage and retrieval of private building block models and private partial system designs.

LIBRARY SYSTEM is still in an early stage of development. At present we use basic facilities of the

TABLE II (Continued)

TYPE	GRAPHICAL	MACHINE PROCESSABLE
A NAMED CONTROLLED DATA PROCESSOR REPRESENTS A DATA TRANSFORMATION OBJECT WHICH IS ACTIVATED WHEN AN ASSOCIATED CONTROL NODE IS INITIATED. E.G., PROCESSOR P1 IS INITIATED WHENEVER EITHER N1 OR N2 IS INITIATED. WHEN PROCESSOR P1 TERMINATES IT CAUSES TOKENS TO BE PLACED ON OUTPUT ARCS OF THE CONTROL NODE WHICH INITIATED IT. AN INTERPRETATION OF THE DATA TRANSFORMATION AND OTHER PARAMETERS SUCH AS TIME DELAY OR RESOURCE REQUIREMENTS CAN BE ASSOCIATED WITH THE DATA PROCESSOR. EXAMPLE: PROCESSOR P1 HAS A RANDOM DELAY ASSOCIATED WITH IT.		@DATA GRAPH; @PROCESSOR P1 (N1,N2); @END; PLIP INTERPRETATION @PROCESSOR P1; DCL IRAND ENTRY(FIXED BIN(31) FIXED BIN(31))RETURNS (FIXED BIN(31)); /*RANDOM # GENERATOR*/ DCL NUMBER FIXED BIN(31); NUMBER=IRAND(1,2) /*PICK AN INTEGER: 1 OR 2*/ IF NUMBER=1 THEN @OUTPUT_ARCS='B1,B2'; ELSE @OUTPUT_ARCS='B3,B4'; @DELAY=IRAND (10,100); /*PICK RANDOM DELAY FROM 10 TO 100*/ @END PROCESSOR;
A NAMED UNCONTROLLED DATA PROCESSOR REPRESENTS A DATA TRANSFORMER WHICH PROVIDES, AT ITS OUTPUT, STATED FUNCTIONS OF ITS INPUTS INDEPENDENT OF CONTROL NODE STATES. IN THE DATA GRAPH AN UNCONTROLLED PROCESSOR IS IDENTIFIED BY PROVIDING AN EXPLICIT DECLARATION. AN INTERPRETATION OF THE DATA TRANSFORMATIONS AND OTHER PARAMETERS MAY BE ASSOCIATED WITH IT IN AN IDENTICAL MANNER TO THE CONTROLLED PROCESSOR.		@DATA_GRAPH; @UNCONTROLLED_PROCESSORS U1; @END;
A NAMED DATA SET REPRESENTS A PASSIVE COLLECTION OF DATA. DATA STRUCTURE MAY BE ASSOCIATED WITH A DATASET. ALL PL/1 DECLARATIONS NOT CONTAINING SCOPE OR STORAGE CLASS ATTRIBUTES ARE ACCEPTED AS DEFINITIONS OF DATA SETS. CHARACTER STRINGS CANNOT HAVE THE VARYING ATTRIBUTE. EXAMPLE: THE DATASET D1 IS A SIX-DECIMAL-DIGIT COMPLEX FLOATING POINT NUMBER.		@DATA_GRAPH; @DATASETS D1; @END; PLIP INTERPRETATION @DATASET D1 COMPLEX FLOAT DECIMAL(6);
A NAMED DATA ARC STATICALLY BINDS DATA PROCESSORS AND DATASETS. A DATA PROCESSOR HAS READ OR WRITE ACCESS TO A DATA SET IF THE ARROW POINTS TO OR FROM THE DATA PROCESSOR RESPECTIVELY. EXAMPLE: PROCESSOR P1 IS INITIATED BY CONTROL NODE N1. P1 READS DATA FROM DATASETS D2 AND D3 AND WRITES THEIR SUM INTO DATASET D1.		@DATA_GRAPH; @PROCESSORS P1(N1); @DATASETS D1, D2, D3; @ARCS DA1, DA2, DA3; DA3 (D3:P1); DA2 (D2:P1); DA1 (P1:D1); @END; PLIP INTERPRETATION @DATASET D1 FIXED BIN(31); @DATASET D2 FIXED BIN(31); @DATASET D3 FIXED BIN(31); @PROCESSOR P1; @READ(D3); @READ(D2); D1 = D2+D3; @WRITE(D1); @END PROCESSOR;

interactive operating systems. There has been a great deal of design thinking and actual DBMS design but the results are too speculative to include here.

Behavioral—Structural mapping

A fundamental aspect of the SARA system is the mapping of behavior on the structural primitives i.e., modules, sockets, and interconnections. The mapping funcitons are currently under development. It is only after such mapping that SARA tools can detect enought cases of inconsistency and incompleteness to repay the designers for the structured thinking and excess description that they are forced to provide when they follow the prescribed methodology. Behavioral attributes are mapped to sockets[5] and focus the designers attention on interface problems, the source of most design errors. Then if two sockets are linked via an interconnection and an inconsistency in attributes is discovered, it is known that either an error was made or else the interaction must have a more complex behavior designed into it.

Multilevel modeling

As indicated in the first part of this paper the strength of support for multilevel modeling procedures and the consequent management of complexity will be the principal test of SARA's reason for existence. During the partition phase, the role of SARA is to support evaluation of a refinement proposed by the designer and to accept it if the higher level abstraction is not contradicted. If the designer insists on the refinement, despite contradiction, then a procedure to weaken the abstraction is needed. Similarly in the composition phase, the role of SARA is to compose given building blocks into a model and to help discover inconsistencies with respect to the next higher level model.

The behavioral models are clearly more complex. Campos and Ruggerio[22] have proposed formal replacement algorithms which retain equivalencies between two graph models of behavior. Whenever a single control or data graph primitive is to be refined or whenever a sub-graph can be abstracted by a single primitive, the replacement algorithm guarantees that changes are localized. In general a refine-

SL1 CODE

```
S2(S3,S1); S2(S3@L2,S1@L2); S2.L2+(S3@L2,S1@L2);
S3(S4,S4@L2); S3.L2(S4@L2,S3@L2);
S1(SUB1,SUB2,SUB3,SUB1@L2,SUB1@L1,SUB2@L1,SUB3@L1);
S1.L1+(SUB1@L1,SUB2@L1,SUB3@L1);
S1.L2+(SUB1@L2,S1@L2);
```

NOTE: THE "." IS USED TO DESCRIBE NESTING IN A FULLY QUALIFIED NAME, E.G. S2.S1.SUB1 IS THE FULLY QUALIFIED NAME OF SUB1 IN FIGURE 6.

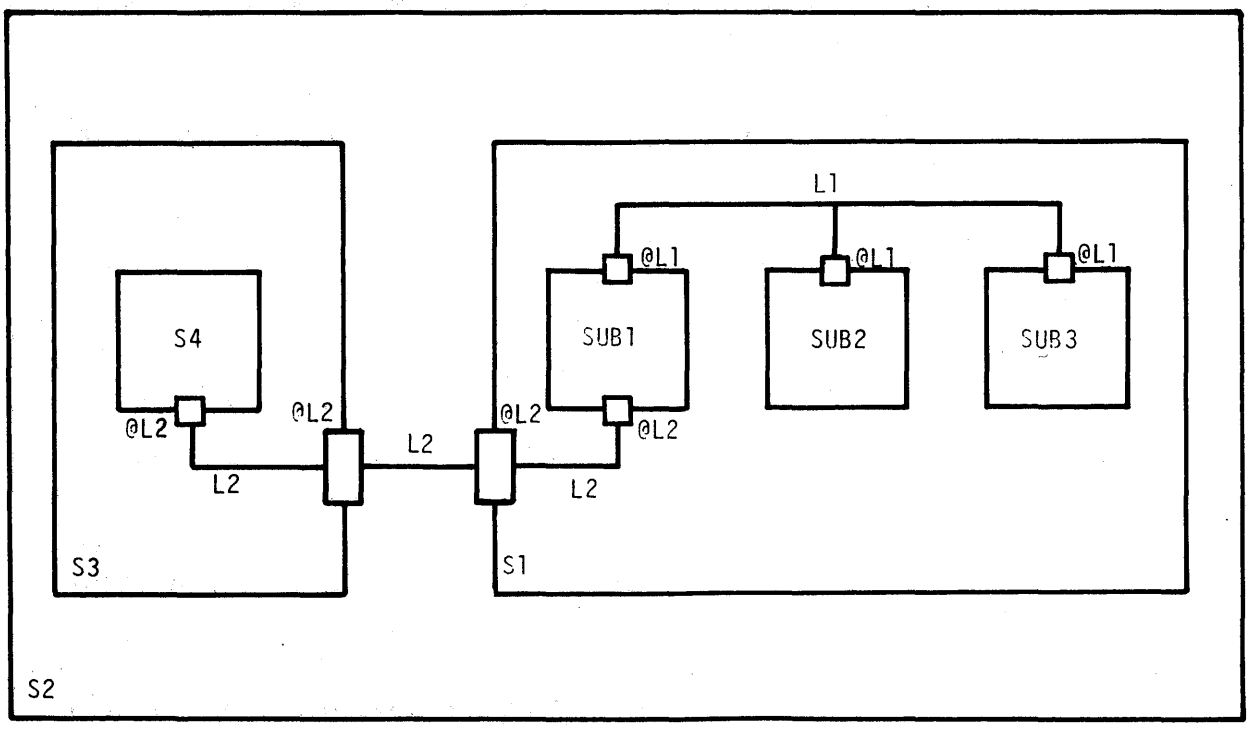


Figure 6—Multi-level modules

ment of (or abstraction into) a data graph primitive implies modification of the associated control graph—and vice versa. Furthermore it is possible to retain "well-behaved" properties like proper termination during these replacements. These replacement algorithms are restricted in the sense that we do not have an algorithm yet to permit replacement of an arbitrary sub-graph by another sub-graph. However they add considerable strength to SARA in support of the designer.

In the same vein, our models of concurrent systems provide a context in which to embed verified processor interpretations and seek algorithms to prove partial correctness of a full data graph including interpretation.

For effective multilevel modeling, the "bottom line" is the accuracy, clarity and richness of the building block models available to the designer. No matter how late or early implementation decisions are explicitly made, the design process must be goal oriented and the designer must either understand the elements or have faith in the models reinforced by successful use of them. For hardware building block models, we find ourselves giving a great deal of attention to detail in order to convince ourselves that our models can accurately represent manufacturers specifications.[23] Drobman[24] has been able to generalize such study by defining "generic" as well as specific building blocks. There are other ways to harness rich populations of building blocks.

SL1 CODE - HIGH LEVEL MODEL

```
UNIVERSE (SUB1,SUB2,SUB3);
UNIVERSE (SUB1@L1,SUB2@L1,SUB3@L1);
L1 + (SUB1@L1,SUB2@L1,SUB3@L1);
```

SL1 CODE - REFINED MODEL

```
SUB1@L1 (SUB1@L1.WIRE1,SUB1@L1.WIRE2,SUB1@L1.WIRE3);
SVB2@L1 (SUB2@L1.WIRE3,SUB2@L1.WIRE2);
SUB3@L1 (SUB3@L1.WIRE2,SUB3@L1.WIRE1);
L1.WIRE1 + (SUB1@L1.WIRE1,SUB3@L1.WIRE);
L1.WIRE2 + (SUB1@L1.WIRE2,SUB2.@L1.WIRE2,SUB3.@L1.WIRE2);
L1.WIRE3 + (SUB1@L1.WIRE3,SUB2@L1.WIRE3);
```

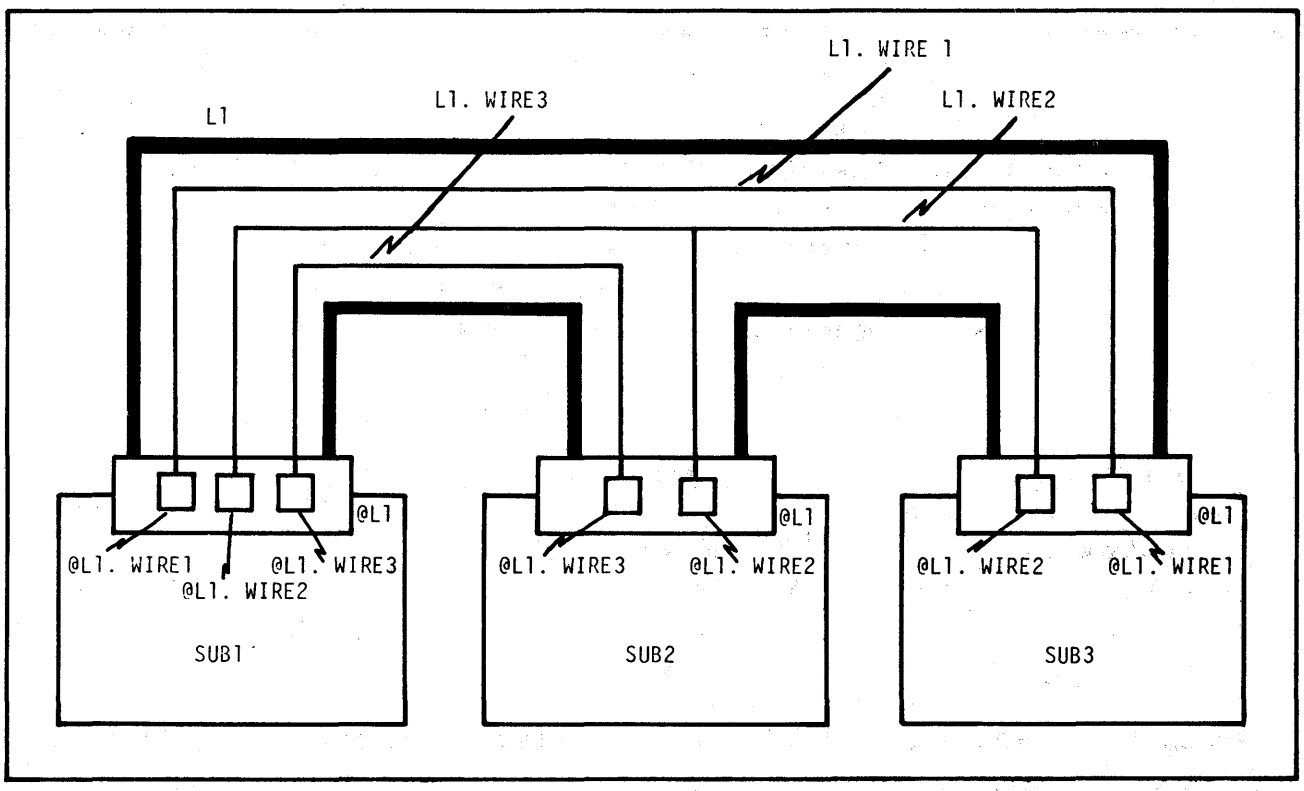


Figure 7—Multi-level interconnection

For example we have a path for translating ISP descriptions into PLIP interpretations but we do not yet fully understand limitations imposed by efficiency issues.

In the companion paper,[5] a software example is taken all the way from programming-in-the-large to programming-in-the-small or PL1 code. In that case the PL1 language definition provides the building blocks and the transition is smooth because the entire SARA System is written in PL1. When concurrency exists, the transition is not completely smooth because the "token machine" is an integral part of the SARA environment and there must be equivalent machinery wherever a SARA model is expected to execute. In fact we are exploring new architectures in which SARA models would run with no alteration of the designed and tested model.

SUMMARY AND CONCLUSION

The UCLA design methodology has been elaborated in an attempt to indicate its requirement-driven and self-documenting properties for modular, concurrent, hardware and software systems. The need for implementation in hardware, for example, may show up explicitly in initialization or finally appear in composition.

Those goal characteristics which distinguish this design philosophy are:

- Explicit modeling of environments using the same primitives as the systems being designed.
- Separate but related static and dynamic models.
- Separate but related control and data flow models.

- Separate but related structural and behavioral models.
- Support for both top-down and bottom-up design.
- Implementation decisions forced in composition but retained when they are entered earlier.
- Supported "well-behaved" properties in multilevel modeling and analysis.
- Supported simulation of concurrent processes.
- Supported fabrication.

SARA is useful now as a collection of tools which help structured design of concurrent systems. SARA has been tested in a number of design problems. As a result of its recent move to MIT-MULTICS, it is soon to be tested on larger system problems. To the extent that support of the requirement-driven multilevel design methodology is increased and detected "errors" are reported to the designer, SARA's value will increase significantly. A number of problems with the current methodology and its support are discussed in the companion paper.[5]

We believe that SARA is unique in the support it gives to design of concurrent systems. We do not look on this methodology as being in competition with other active work in, for example, dataflow architecture or sequential software design. We are constantly alert to incorporation of other maturing synthesis techniques, particularly when any of our designs reach the stage of incorporating sequential programs.

The number of poorly understood problems which have been pointed to in this paper is very large. They give credence to the author's philosophy which rejects completely-automatic design at this time and seeks to support the designers injection of value judgments and decisions. Automated design procedures should be introduced selectively and patiently. Sarah begat Isaac when she was ninety years old [Genesis]. It is possible that, with the help of SARA and the help and the wisdom of many researchers and machines, we will know enough before 2067 to beget a powerful ISAAC (Information System Automatic Architect Computer). We do know how to beget an ISAAC but also know that a 1978 ISAAC would be fired for incompetence long before tenure could be achieved.

REFERENCES

1. Dahl, C. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming*, Academic Press, London, 1972.
2. Liskov, B. E. and S. N. Zilles "Programming with Abstract Data Types," *Proceedings of a Symposium on Very High Level Languages*, in *SIGPLAN Notices* 9:4, April 1974, pp. 50-59.
3. Wulf, W. A., "ALPHARD: Toward a Language to Support Structured Programs," Technical Report, Carnegie-Mellon University, April 1974.
4. Popek, G., J. J. Horning, B. W. Lampson, R. London, J. Mitchell, "Notes on the Design of Euclid," Raleigh, North Carolina, March 28-30, 1978.
5. Campos, Ivan M. and G. Estrin, "SARA Aided Design of Software for Concurrent Systems," *AFIPS Conference Proceedings*, Volume 47, 1978 National Computer Conference.
6. *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA., Feb. 24-25, 1977.
7. Robinson, Lawrence and Karl N. Levitt, Proof Techniques for Hierarchically Structured Programs, *Communications of the ACM*, April 1977, Volume 20, No. 4.
8. Stewart, Barbara and C. Herman, "Design Analysis: An Approach for Improving the System Design Process," M.S. in Engineering, University of California, Los Angeles, CA., 1976.
9. Boehm, Barry, W., "Software Engineering: R & D Trends and Defense Needs," *Proceedings of the Conference on Research Directions in Software Technology*, Oct. 10-12, 1977.
10. Gostelow, K., V. Cerf, G. Estrin, and S. Yolansky, "Proper Termination of Flow-of-Control in Programs Involving Concurrent Processes," *Proceedings of the ACM*, Vol. 11, Boston, August 1972, pp. 742-754.
11. Overman, William T., "Formal Verification of GMBs," Computer Science Department, University of California, Los Angeles, CA., Internal Memorandum #176, July 1977.
12. Parnas, D. I., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.
13. Gardner, R. I., "A Methodology for Digital System Design Based on Structural and Funcitonal Modeling," Technical Report, UCLA-ENG-7488, January 1975.
14. Fenchel, R., "SARA User's Manual System ARchitects Apprentice," Computer Science Department, University of California, Los Angeles, CA., January 11, 1977.
15. Overman, William T., "PLIP System Reference Manual," Computer Science Department, University of California, Los Angeles, July 29, 1977.
16. Razouk, R. and G. Estrin, "The Graph Model of Behavior Simulator," *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA., February 24-25, 1977.
17. Potash, H., A. Tyrill, D. Allen, S. Joseph and G. Estrin, "DCDS Digital Simulating System," *AFIPS Conference Proceedings*, Vol. 35, Fall Joint Computer Conference, 1969.
18. Bell, Gordon and A. Newell, *Computer Structures: Readings and Examples,"* McGraw-Hill Book Company, 1971.
19. Crocker, Stephen D., "State Deltas: A Formalism for Representing Segments of Computation," Ph.D. in Computer Science, University of California, Los Angeles, August 1977.
20. Knuth, D. and J. McNeeley, "SOL—A Symbolic Language for General-Purpose Systems Simulation," *IEEE Transactions on Computers*, August 1964, pp. 401-408.
21. Ulfers, H. E., "Packnet—A Packet Switched Data Network Simulator," International Conference on Communications, 1975.
22. Campos, Ivan M., "Multilevel Modeling for Synthesis of Reliable Concurrent Software Systems," Ph.D. in Computer Science, University of California, Los Angeles, CA., November 1977.
23. Overman, William and G. Estrin, "Developing A SARA Building Block—The "8080," *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA., Feb. 24-25, 1977.
24. Cerf, V., B. Choi and C. Maxwell, "A Reduction Algorithm Implementation," Internal Memorandum #108, Digital Technology Research Group, Computer Science Department, University of California, Los Angeles, September 1972.
25. Drobman, Jeffrey H., "Building Block Modeling Methodology for Composition of Microprocessor Based Digital Systems," Ph.D. Dissertation, Computer Science, University of California, Los Angeles, in progress.
26. Estrin, Gerald, "Modeling for Synthesis—The Gap Between Intent and Behavior," *Proceedings of Symposium on Design Automation and Microprocessors*, Palo Alto, CA., Feb. 1977.
27. Fenchel, Robert, "A System Control Processor for Microcomputer Network," June 1976, UCLA-ENG-7646.

SARA aided design of software for concurrent systems*

by IVAN M. CAMPOS** and GERALD ESTRIN
University of California at Los Angeles
Los Angeles, California

INTRODUCTION

In the past several years it has often been noted that software is the major cost component in designing, implementing and maintaining computer systems. Unreliability and associated debugging and maintenance activities are among the main elements responsible for this increase in cost. Most bugs are "born" in the design phase, i.e., the design itself is usually faulty,[1] and when a bug is uncovered some changes are likely to occur. Whenever modification is made, it is generally difficult to keep track of all consequences that it may provoke. Poor design often shows up as a high degree of connectivity among subsystems.[2,3] Interconnections may be "hidden," representing unstated assumptions that a subsystem makes about other subsystems and leading to unforeseen side effects when seemingly harmless changes are made.[4,5]

In an attempt to improve software, extra effort tends to be invested in design. Considerable attention is being given to issues such as stepwise refinement,[6,7] structured programming,[8] modularity,[9,10] and composite design.[3] Data abstraction concepts have been introduced into languages such as CLU,[11] EUCLID,[12] and ALPHARD.[13] Active research continues to uncover more effective methods for analysis and synthesis of verified programs.[14]

Since the early 1960's, Estrin and his colleagues have been involved in the development of models of computational algorithms, models of interpreters, and procedures for mapping one on the other. The UCLA Graph Model of Computation;[15–21] the development of a methodology (DCDS) for design of hierarchical hardware and firmware systems;[22] and the revelation of difficulties in evaluation of unstructured computer systems[23] led this UCLA group to concentrate on synthesis of structured systems, in which one might manage complexity from the outset and prepare for further analysis.

The major direction of work at UCLA on methodology of synthesis is to complete a set of tools which support a structured multilevel design procedure for software or hardware development.[24] This interactive computer-aided system called SARA (Systems ARchitect's Apprentice) provides languages to help a designer form useful abstractions which can be manipulated and tested in a disciplined way,[25–28] and which may be forced to retain good properties.[29]

SARA supports both a bottom-up composition (abstraction) procedure and a top-down partitioning (refinement) procedure. In both cases, however, the procedures are requirement-driven so that attention can be focused on the gap between a designer's declared intent and the current approximation of the system's behavior.

Models are created using a small number of primitives* whose manipulation are supported by SARA interactive translators and a simulator (see Tables I and II in the companion paper[24]).

Modules, sockets and interconnections are structural modeling primitives which allow creation of a hierarchical name space for a fully nested system. Named modules permit interaction only through sockets. Socket names are known both inside and outside of modules, thus supporting declarations of interfaces. Named interconnections support interaction among modules.

Explicit control flow behavior can be expressed by control nodes, which span initiation and termination of associated processes, and by directed control arcs, which are used to express precedence conditions. Control conditions and thresholds can be prescribed to condition process initiation. Control consequences can be prescribed to take effect following process completion. Tokens mark the state of control flow and a token machine governs progress of the state of a model. Data flow is described by graphs composed of controlled processors, uncontrolled processors, datasets and data arcs.

All of the above primitives are currently used to form models which are static in the sense that the topology currently does not change during execution. Interpretations can be associated with the data flow primitives (processors and datasets) and provide the semantics for modeling of dynamics. Currently, a pre-processed PL/I (PLIP) is the interpretation language. An interactive simulator supports experi-

* This research was supported by the U.S. Department of Energy, Cont No. EY-76-S-03-0034, PA214.

** Presently at: Ivan Moura Campos, Departamento De Ciencia Da Computacao, Universidade Federal De Minas Gerais, Pampulha, Belo Horizonte—MG, 30 000 Brasil

* The authors chose not to provide, in this paper, the full detail of syntax and semantics implemented in SARA. The interested reader can find such information in Refernce 26. We have tried to include enough detail to follow examples. Note that the "@" symbol is used for many purposes: to identify a socket name or a declaration or language processor.

ments on models. Extensive discussion of SARA modeling may be found in the references noted above.

The objective of this paper is to expose the SARA methodology as applied to software. The central problem is the set of tools and procedures to be brought to bear on building possibly concurrent software systems which reliably carry out the designer's intent.

We also wish to propose a role of SARA models not only as design and evaluation tools, but also as the basis for a superstructure to be overlaid on the actual code of a designed product as redundant information for the operating system. One consequence of imposing such a superstructure on source code modules is that it provides its operating system and its user with a "live" documentation whose objective is to enforce consistency between requirements, structure, function and behavior of the software system.

In the body of the paper below, we take the reader through the key steps of two example designs and try to generalize the procedure through commentary. We chose the design of a synchronized sender-receiver process and an abstracted FFT. They are relatively simple examples which are of some importance and serve to bring out many properties of our design method and the SARA system.

Following an informal statement of need, the design universe is partitioned into the system to be designed and its environment. An initial behavioral model (of the system and of the environment) fixes all assumptions under which the design is to proceed. Requirements establish evaluation criteria for the system. Following "top-down" levels of designer-created behavior, there is a final composition step to produce PL/I code. In the models below, comments will highlight SARA conventions. In all other cases PL/I conventions will be assumed to hold.

A DESIGN EXAMPLE—A SIMPLE MESSAGE TRANSMITTER

The informal statement of our design problem is "design a system whereby a process produces and sends a sequence of data to another process, if the latter is not busy.

DEFINING THE STRUCTURE

The initial design steps in the SARA design procedure[24,27] lead us to partition our UNIVERSE into two SL1 modules, MESSAGE TRANSMITTER, the system to be designed, and ENV, its environment. The machine processable SL1 code, its graphical representation and the requirements for both ENV and MESSAGE TRANSMITTER are shown in Figure 1. Parentheses establish nesting level so that the second line of SL1 code is read "ENV" is composed of three sockets, called @INFILE, @HERALD, and @OUTFILE." Identifiers starting with the character '@' are socket names. The symbol '+' indicates that the system to its left is an interconnection which connects the sockets whose names appear between the parentheses that follow it. Thus the third SL1 statement says that "within UNIVERSE there is an interconnection named SL1 which connects the socket ENV @INFILE to MESSAGE TRANSMITTER @INFILE."

The three interfaces between the modules that constitute ENV and MESSAGE TRANSMITTER are called HERALD, INFILE and OUTFILE, respectively, as shown in the sockets.

SL1 CODE FOR STRUCTURAL MODEL OF UNIVERSE

```
UNIVERSE(ENV,MESSAGE_TRANSMITTER);
ENV(@INFILE,@HERALD,@OUTFILE);MESSAGE_TRANSMITTER(@INFILE,@HERALD,@OUTFILE) ;
UNIVERSE(L1+(ENV@INFILE,MESSAGE_TRANSMITTER@INFILE));
UNIVERSE(L2+(ENV@HERALD,MESSAGE_TRANSMITTER@HERALD));
UNIVERSE(L3+(ENV@OUTFILE,MESSAGE_TRANSMITTER@OUTFILE));
```

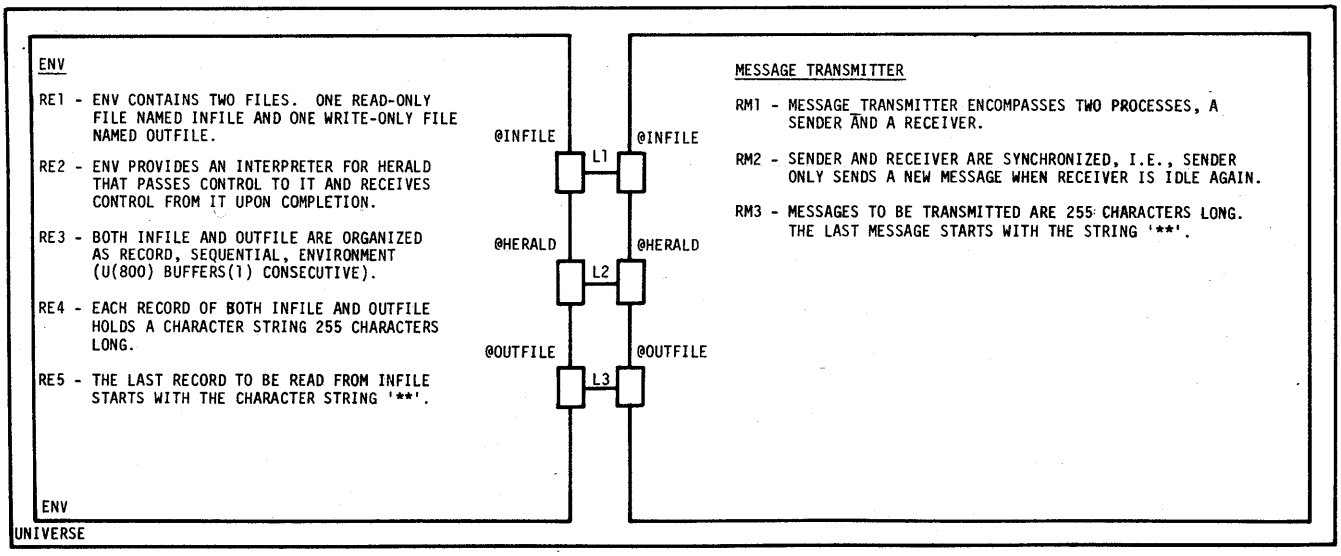


Figure 1—Requirements superimposed on ENV and message_transmitter

THE BEHAVIORAL MODEL GMB (GRAPH MODEL OF BEHAVIOR)

Figure 2 contains the graphical representation of the GMB model of the interaction between ENV and MESSAGE TRANSMITTER, its machine processable representation (upper left), the PLIP interpretation associated with the datasets and the processor of the data graph (upper right), the explicit mapping of GMB primitives into the structure (lower left) and the list of attributes for each socket (lower right). Notice that the correspondence between node N2 and processor P2 is indicated in the source GMB code in both the "@nodes" statement and the "@processor" statement. The initial state of the graph is shown to consist of a single token

DESCRIPTION OF UNINTERPRETED GMB

```
@CONTROL_GRAPH
  @NODES N1, N2(P2), N3;
  @ARCS S(1), A1, A2, X;
  N1(S:A1);
  N2(A1:A2);
  N3(A2:X);
@END;
@DATAGRAPH;
  @PROCESSORS P2(N2);
  @DATASETS INFILE, OUTFILE;
  @ARCS DA1, DA2;
  DA1(INFILE:P2);
  DA2(P2:OUTFILE);
@END;
```

INTERPRETATION OF DATA GRAPH

PLIP

```
@DATASET INFILE FILE RECORD SEQUENTIAL INPUT
   ENVIRONMENT(U(800) BUFFERS(1) CONSECUTIVE);
@DATASET OUTFILE FILE RECORD SEQUENTIAL OUTPUT
   ENVIRONMENT (U(800) BUFFERS(1) CONSECUTIVE);
@PROCESSOR P2;
  DCL MESSAGE CHAR (255);
  OPEN FILE(INFILE);
  OPEN FILE(OUTFILE);
  /* SENDER PROCESS */
  LOOP:  READ FILE(INFILE) INTO(MESSAGE);
  @READ(INFILE);
  /* RECEIVER PROCESS */
  WRITE FILE(OUTFILE) FROM(MESSAGE);
  @WRITE(OUTFILE);
  /* TEST FOR COMPLETION */
  IF SUBSTR(MESSAGE,1,2)¬= '**'
     THEN GO TO LOOP;
  CLOSE FILE(INFILE);
  CLOSE FILE(OUTFILE);
@ENDPROCESSOR;
```

GMB TO SL1 MAPPING

GMB PRIMITIVE		SL1 PRIMITIVE	
NAME	TYPE	NAME	TYPE
S	C_ARC	ENV	MODULE
N1	C_NODE	ENV	MODULE
A1	C_ARC	ENV@HERALD	SOCKET
A2	C_ARC	ENV@HERALD	SOCKET
N3	C_NODE	ENV	MODULE
X	C_ARC	ENV	MODULE
INFILE	DATASET	ENV	MODULE
DA1	D_ARC	ENV@INFILE	SOCKET
DA2	D_ARC	ENV@OUTFILE	SOCKET
OUTFILE	DATASET	ENV	MODULE
A1	C_ARC	L2	INTERC
A2	C_ARC	L2	INTERC
DA1	D_ARC	L1	INTERC
DA2	D_ARC	L3	INTERC
A1	C_ARC	MESSAGE_TRANSMITTER@HERALD	SOCKET
A2	C_ARC	MESSAGE_TRANSMITTER@HERALD	SOCKET
N2	C_NODE	MESSAGE_TRANSMITTER	MODULE
DA1	D_ARC	MESSAGE_TRANSMITTER@INFILE	SOCKET
P2	PROCESSOR	MESSAGE_TRANSMITTER	MODULE
DA2	D_ARC	MESSAGE_TRANSMITTER@OUTFILE	SOCKET

SOCKET ATTRIBUTES

SL1 SOCKET	ATTRIBUTES
ENV@INFILE	DATA OUTPUT CHARACTER (255) NONVARYING UNALIGNED STATIC EXTERNAL
ENV@HERALD	CONTROL INPUT-OUTPUT
ENV@OUTFILE	DATA INPUT CHARACTER (255) NONVARYING UNALIGNED STATIC EXTERNAL
MESSAGE_TRANSMITTER@INFILE	DATA INPUT CHARACTER (255) NONVARYING UNALIGNED STATIC EXTERNAL
MESSAGE_TRANSMITTER@HERALD	CONTROL INPUT-OUTPUT
MESSAGE_TRANSMITTER@OUTFILE	DATA OUTPUT CHARACTER (255) NONVARYING UNALIGNED STATIC EXTERNAL

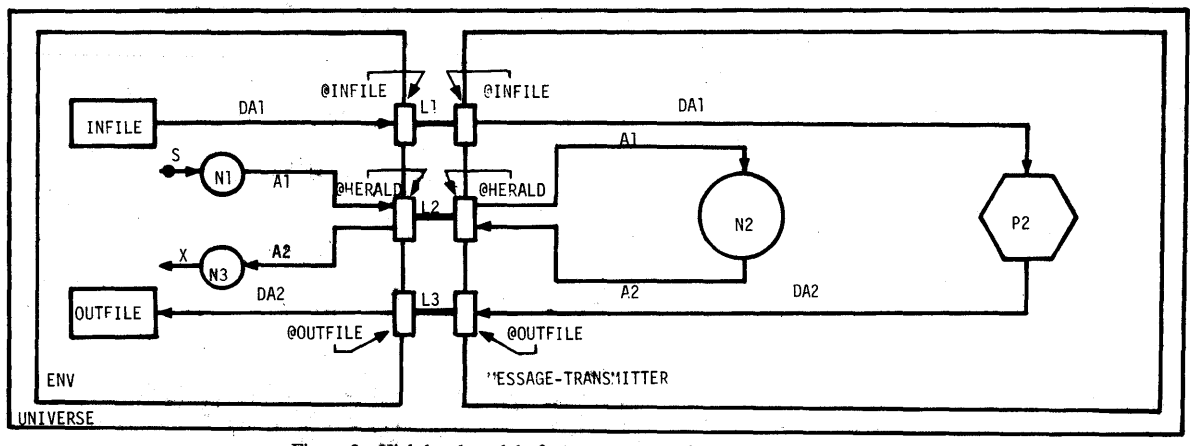


Figure 2—High level model of message_transmitter and its environment

on arc S. The expression in parentheses after node names contain arc expressions separated by a colon. To the left of the colon one specifies the input logic expression to the node (precondition for its activation); conversely to the right of the colon one specifies the output logic expression (postcondition to its deactivation). Any well-formed logic expression can be used. If the designer is restricted to using the operators AND ("*") and OR ("+") then analysis of control graph properties is made easier. Let us now consider evaluation of the model with respect to the requirements. RE1 and RE3 are satisfied by the first two PLIP dataset interpretations. RE2 is satisfied by the control graph description. RE4 is satisfied by the first DCL in the process P2 interpretation. RE5 is implied by the interpretation of processor P2. RM1 is satisfied by inspection. RM2 is satisfied because the Sender and Receiver processes are done sequentially. RM3 can be tested in the SARA simulator environment and, finally, on a PL1 machine when the program is composed.

The above model is read by the SARA translator and simulated, even at this high level, by providing test messages in INFILE and observing at the end of simulation that they appear in OUTFILE in a manner that satisfies the requirements.

CODE SYNTHESIS—BOTTOM-UP COMPOSITION

This design example is so simple that we directly go bottom-up in the design process.

A few observations are in order. The fact that PLIP is an extension to PL/I via a preprocessor places us in a privileged position to directly synthesize PL/I code. We will try to treat each PLIP statement as a building block model of the corresponding PL/I statement. We assume that we have validated MESSAGE TRANSMITTER with respect to requirements and are therefore ready to directly synthesize the code. This can be seen as a "fabrication phase" after the modeling has been completed, and we assume that future development of SARA might make this process automatic.

We compose the MESSAGE TRANSMITTER procedure by declaring as internal variables all those datasets that have been mapped separately on ENV in the GMB-to-SL1 mapping (in this case INFILE and OUTFILE). We concatenate them with the declarations of the internal variables found in the PLIP code of P2.

The next "cleaning-up" step consists of deleting all the @READ and @WRITE statements from the code, since they are only used by the SARA GMB simulator. The final code is shown in Figure 3 next to the original PLIP code (taken from the upper right box in Figure 2).

This was, of course, a trivial example but served to illustrate the methodology. Our second design example is more complex.

A SECOND DESIGN EXAMPLE—A FAST FOURIER TRANSFORMER[30]

In this example we wish to focus the readers attention on the composition procedure, which is our name for a bottom-up design procedure.

A signal processing problem which has received extensive attention can be informally stated as "design a system which receives sequences of complex numbers; calculates the Fast Fourier Transform of each of those sequences; and makes it available to the environment that issued the original complex numbers." For purposes of this paper we consider a reduced problem for sequences of 8 complex numbers. It is of some importance that the reader extrapolate the application of the method to larger problems.

In this design example we get a little more formal in our discussion. We use a specification of the problem (which we call our invariant) whereby a description of the desired in-

INTERPRETATION OF DATA GRAPH

PLIP

```
@DATASET INFILE FILE RECORD SEQUENTIAL INPUT
   ENVIRONMENT(U(800) BUFFERS(1) CONSECUTIVE);
@DATASET OUTFILE FILE RECORD SEQUENTIAL OUTPUT
   ENVIRONMENT (U(800) BUFFERS(1) CONSECUTIVE);
@PROCESSOR P2;
   DCL MESSAGE CHAR (255);
   OPEN FILE(INFILE);
   OPEN FILE(OUTFILE);
   /* SENDER PROCESS */
   LOOP:  READ FILE(INFILE) INTO(MESSAGE);
   @READ(INFILE);
   /* RECEIVER PROCESS */
   WRITE FILE(OUTFILE) FROM(MESSAGE);
   @WRITE(OUTFILE);
   /* TEST FOR COMPLETION */
   IF SUBSTR(MESSAGE,1,2)¬= '**'
      THEN GO TO LOOP;
   CLOSE FILE(INFILE);
   CLOSE FILE(OUTFILE);
@ENDPROCESSOR;
```

PL1 CODE

```
MESSAGE_TRANSMITTER:  PROCEDURE
   DCL INFILE FILE RECORD SEQUENTIAL INPUT
   ENVIRONMENT(U(800) BUFFERS(1) CONSECUTIVE);
   DCL OUTFILE FILE RECORD SEQUENTIAL OUTPUT
   ENVIRONMENT(U(800) BUFFERS(1) CONSECUTIVE);
   DCL MESSAGE CHAR (255);
   OPEN FILE(INFILE);
   OPEN FILE(OUTFILE);
   /* SENDER PROCESS */
   LOOP:  READ FILE(INFILE) INTO(MESSAGE);
   /* RECEIVER PROCESS */
   WRITE FILE(OUTFILE) FROM(MESSAGE);
   /* TEST FOR COMPLETION */
   IF SUBSTR(MESSAGE1,2)¬= '**'
      THEN GO TO LOOP;
   CLOSE FILE(INFILE);
   CLOSE FILE(OUTFILE);
END MESSAGE_TRANSMITTER
```

Figure 3—Original PLIP interpretation and final composed PL1 code for message_transmitter

teraction between the system to be designed (FFT) and its environment (ENV) is produced. This invariant is composed of:

- A structural model which splits the design universe into two subsystems, ENV and FFT, and shows their structured connections by means of interconnected sockets.
- A list of requirements for both the system under design and its environment.
- A graph model of behavior (GMB) of the desired interaction between ENV and FFT.
- An interpretation associated with the GMB, which shows in procedural form which input/output transformations are to be performed by both ENV and FFT, with particular attention to interface issues.

THE STRUCTURAL MODEL

The machine processable and graphic SL1 model of our design universe is presented in Figure 4 where we distinguish two subsystems of UNIVERSE, namely ENV and FFT.

Structurally, ENV and FFT are connected through interconnections XI and FI, each of which interconnect a socket in ENV to a socket in FFT. These interconnections support both control and data interactions between the two subsystems. In Figure 4 we have chosen to show both a top level structural model and a more detailed socket model. We are anticipating modeling concurrent programs. Eight of the interconnections support flow from ENV to FFT, and the other 8 support flow from FFT to ENV.

REQUIREMENTS

The requirements are superimposed on the structural model in Figure 4. They are separated into two sets, one mapped onto ENV and the second onto FFT.

THE BEHAVIORAL MODEL

The flow-of-control and the flow-of-data graphs are shown in Figures 5 and 6 respectively, mapped onto the ENV and

TOP LEVEL SL1

```
UNIVERSE (ENV,FFT);
ENV (@XI,@FI);
FFT (@XI,@FI);
UNIVERSE (XI+(ENV@XI,FFT@XI));
UNIVERSE (FI+(ENV@FI,FFT@FI));
```

REFINED SL1

```
UNIVERSE (ENV,FFT);
ENV (@X0,@X1,@X2,@X3,@X4,@X5,@X6,@X7,@F0,@F1,@F2,@F3,@F4,@F5,@F6,@F7);
FFT (@X0,@X1,@X2,@X3,@X4,@X5,@X6,@X7,@F0,@F1,@f2,@F3,@F4,@F5,@F6,@F7);
UNIVERSE (X0+(ENV@X0,FFT@X0), X1+(ENV@X1,FFT@X1), X2+(ENV@X2,FFT@X2), X3+(ENV@X3,FFT@X3),
          X4+(ENV@X4,FFT@X4), X5+(ENV@X5,FFT@X5), X6+(ENV@X6,FFT@X6), X7+(ENV@X7,FFT@X7));
UNIVERSE (F0+(ENV@F0,FFT@F0), F1+(ENV@F1,FFT@F1), F2+(ENV@F2,FFT@F2), F3+(ENV@F3,FFT@F3),
          F4+(ENV@F4,FFT@F4), F5+(ENV@F5,FFT@F5), F6+(ENV@F6,FFT@F6), F7+(ENV@F7,FFT@F7);
```

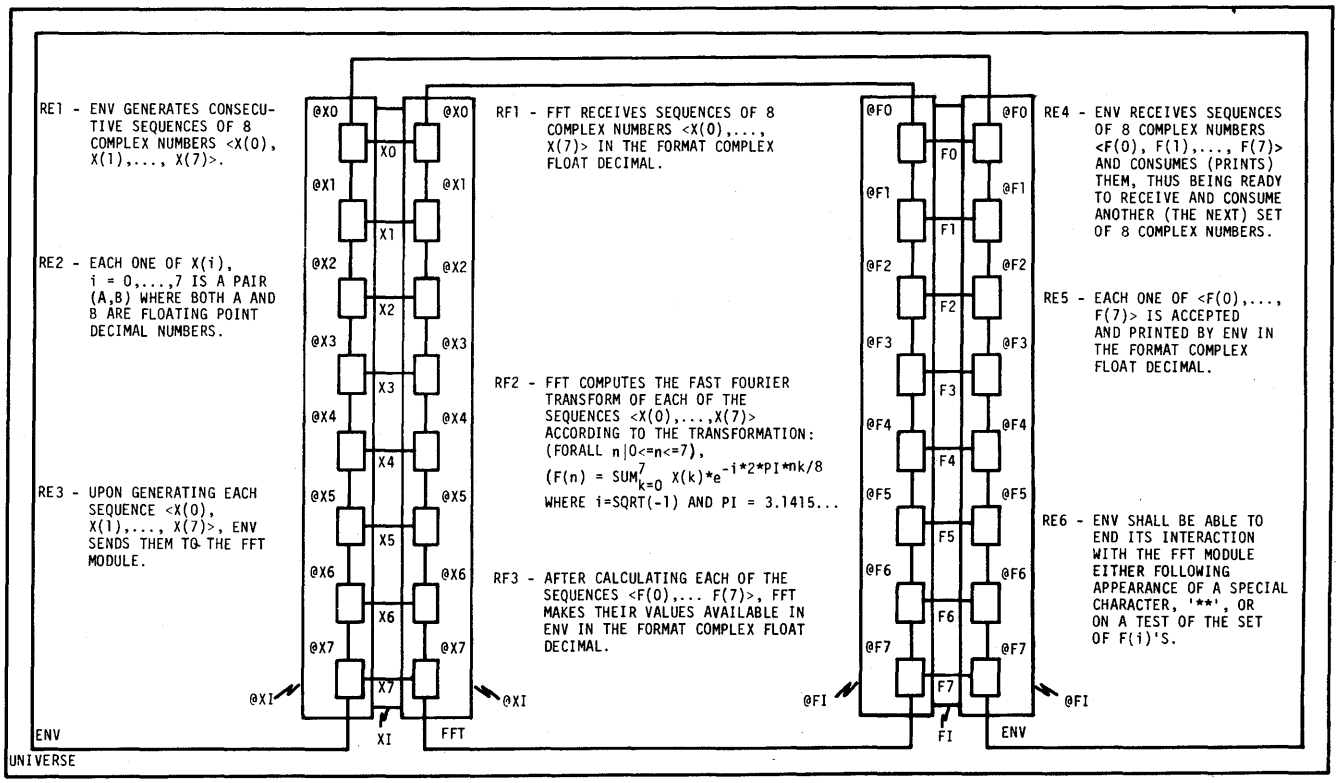


Figure 4—Structural model and requirements for FFT system showing SL1 for both abstract and refined socket levels

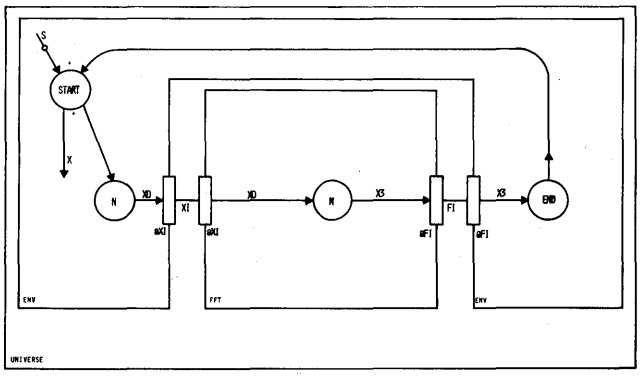


Figure 5(a)—High level FFT control graph

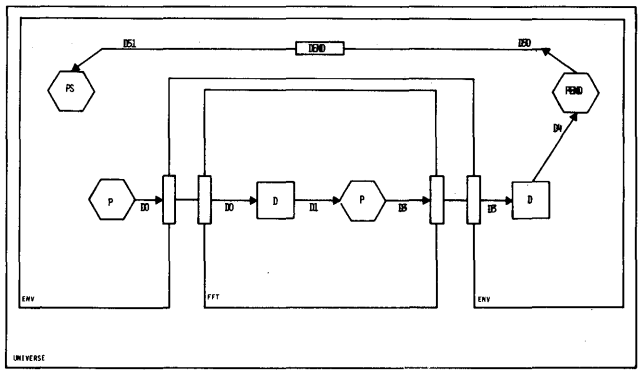


Figure 6(a)—High level FFT data graph

FFT structures. Figures 5A and 6A show the top level models. Figures 5B and 6B show more refined models. The machine processable representation of the GMB, the GMB to SL1 mapping, the PLIP interpretation and the socket attributes are generated in the same manner as for the design example MESSAGE TRANSMITTER. The detail is not included in this paper but is available on line at MIT-Multics. These high level models can be reduced to PL1 code in the same manner as the earlier example.

Let us assume that our tests check that the simulation model is proper and we consider composition utilizing a "butterfly" building block.

THE BUTTERFLY BUILDING BLOCK

The following description should give the reader a firmer idea of what is meant by a software building block in SARA. We continue our discussion with some informality in order to conserve space and to be readable. A building block model includes a name-set, an attribute set, structural models and behavioral models.

The name set provides a place to list all alias library names. In this case we call our building block FFT BUT. However the library might also respond to aliases like FFT BUTTERFLY or FFT BMOD.

The attribute set includes an informal description and comments about interpreters and versions. For this case the attribute set reads:

Informal description

The FFT "butterfly" whose non-procedural model and code are shown below in PL/I-like syntax is a software module designed to run in a multiprocessing environment (concurrently with other such computations) and perform one complex multiplication, one complex addition and one complex subtraction. The latter two operations are done in parallel within the butterfly.

Its usefulness is based on the existence of highly parallel algorithms that can be composed by binding several such procedures together in ingenious ways in order to compute the Fast Fourier Transform of arrays of complex numbers.

The input and output assertions for the Butterfly are

10**(−70)⇐abs(X0)⇐10**70
10**(−70)⇐abs(Y0)⇐10**70
X1=X0+WPY0, where WPY0 is a complex constant
Y1=X0−WPY0, where WPY0 is a complex constant
10**−70⇐abs(WPY0)⇐10**70

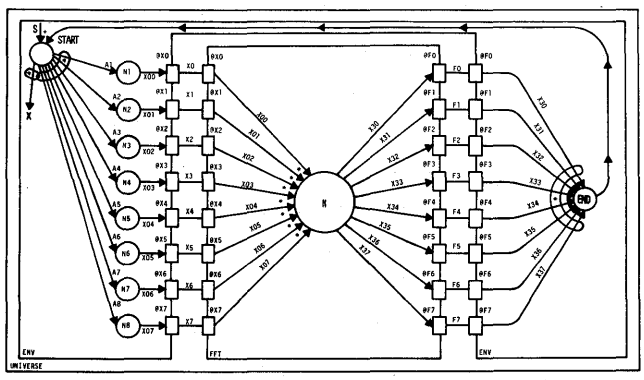


Figure 5(b)—Refined control graph

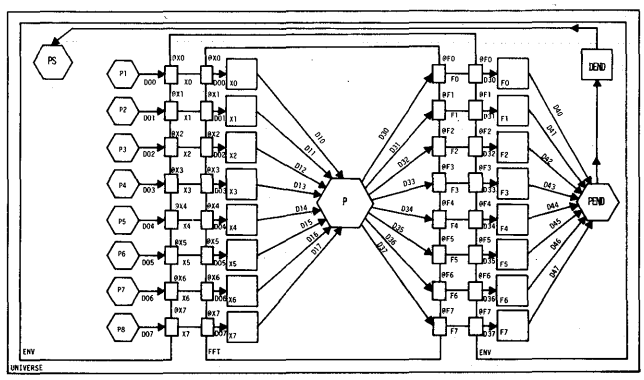


Figure 6(b)—Refined FFT data graph

```
FFT BUT: PROCEDURE;
      DCL (X0, Y0, X1, Y1) COMPLEX DECIMAL
    FLOAT (6)
  EXTERNAL;
      DCL (A3, A4, A7, A8) SEMAPHORE;
  DCL WP COMPLEX DECIMAL FLOAT(6) INITIAL(*);
  DCL WPY0 COMPLEX DECIMAL FLOAT(6);
  .....
  WAIT(A3 & A4)
  WPY0=WP * Y0;
  COBEGIN
  X1=X0+WPY0;
  Y1=X0-WPY0;
  COEND;
  SIGNAL (A7 & A8);

END FFT BUT;
```

Interpreters and versions

This procedure runs in PL/I multitasking environments, provided the availability of the following Concurrent PASCAL[31,32] constructs:

```
cobegin . . . coend
signal (semaphore expression)
wait (semaphore expression)
```

An ingenious implementation of Brinch Hansen's semaphores as an add-on to the PL/I machine is given in Reference 33.

We notice here that the construct cobegin . . . coend can trivially be implemented with semaphores. For each concurrently initiated task we issue a "signal(semaphore)", having a unique semaphore per task. Inside of each of those tasks, the first executable statement is a "wait(semaphore)" which waits for the appropriate semaphore. Upon completion, each task issues a "signal(completion semaphore)", one semaphore per task. The "coend" construct is then substituted by a set of "wait(completion semaphore)".

We shall use the cobegin . . . coend for it is syntactically more pleasing.

The structural and behavioral models are in the same form as used in our top down system design descriptions.

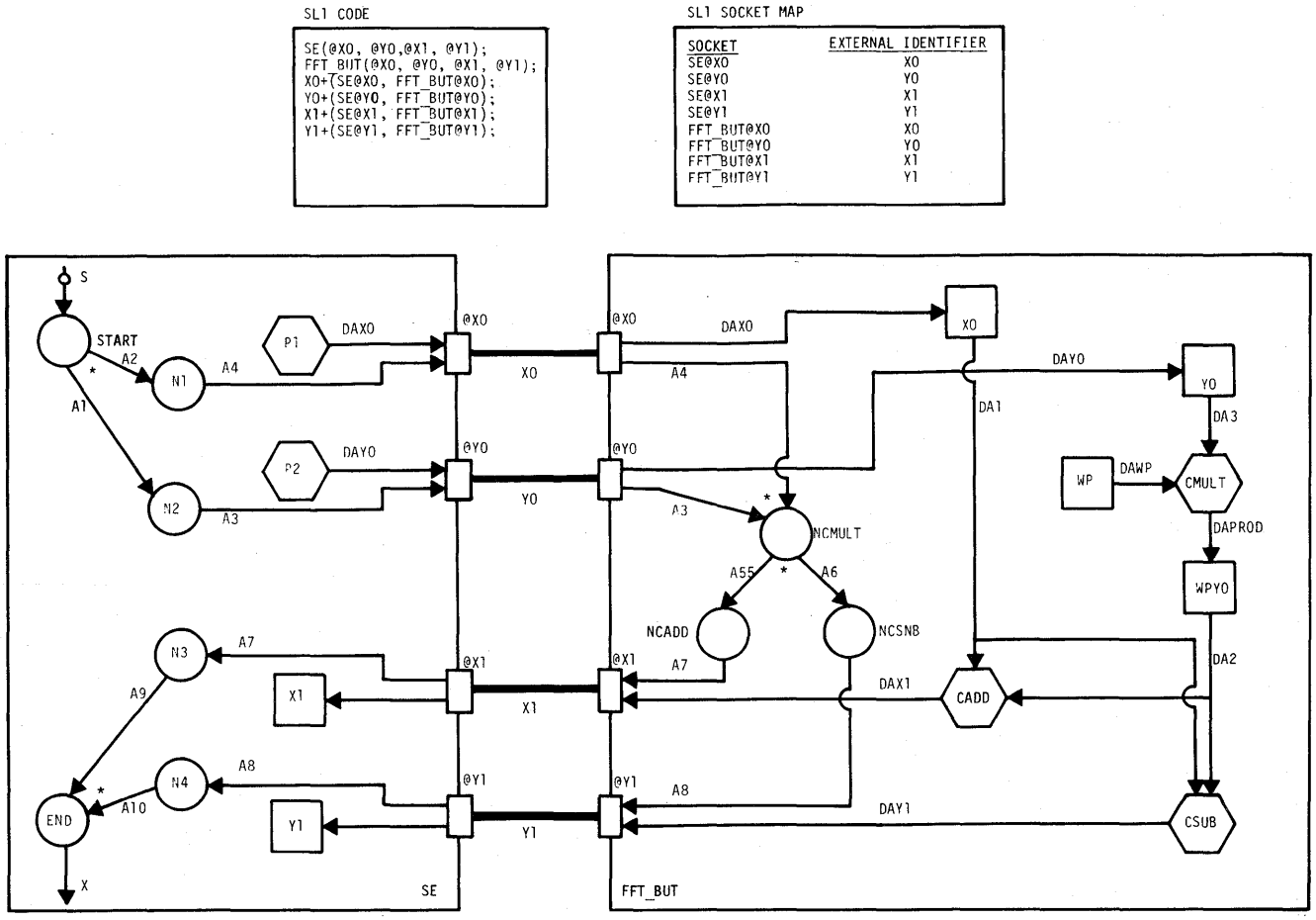


Figure 7—SL1 mapped graph model of FFT "butterfly" building block with its standard environment

MACHINE PROCESSABLE SL1 MODEL

Figure 7 shows a graphical representation of the GMB model of FFT BUT and of its standard environment SE. The structural model SL1 code and the socket to external identifier mapping are included in the figure.

MACHINE PROCESSABLE BEHAVIORAL MODEL

In similar fashion to the MESSAGE TRANSMITTER example above the building block model includes, for it and its standard environment, a control graph description, a data graph description, a PLIP interpretation for each processor and dataset, a socket behavior model and a mapping from GMB to SL1. If space permitted, the data graph interpretations would show how the SE processors P1 and P2 produce the real and complex inputs X0 and Y0 respectively and write them in the named datasets. The initiated multiplication process would be shown to use the input data and be followed by concurrent addition and subtraction. The real and complex results, X1 and Y1 respectively, would be shown to be delivered to the named SE data sets completing the operation. The full model has been executed and checked in the SARA simulator.

Additional behavioral characteristics of the model could be included. A memory requirement model might specify a bound of 1000 bytes. A precision requirement might specify that the outputs have a precision to >5 decimal digits. A timing model might specify a maximum time of one millisecond for each transformation.

COMPOSITION

We shall try now to describe composition of the FFT module using an ensemble of FFT Butterfly building block models to implement the algorithm defined by Cooley and Tukey.[30]

For an n=8 point FFT we have three basic alternatives for arranging the butterflies, in increasing order of parallelism, namely:

(a) one butterfly performing the computation in (n*log(n))/2 (=12) butterfly steps.
(b) n/2 (=4) butterflies performing the computation in 3 butterfly steps.
(c) n/2*log(n) (=12) butterflies performing the computation in one butterfly step.

We chose the third alternative, for it yields the most parallel (and more interesting for our demonstration purposes) algorithm.

At the time of writing this paper the composition procedure for taking building block models out of SARA's library and linking them together automatically had not yet been

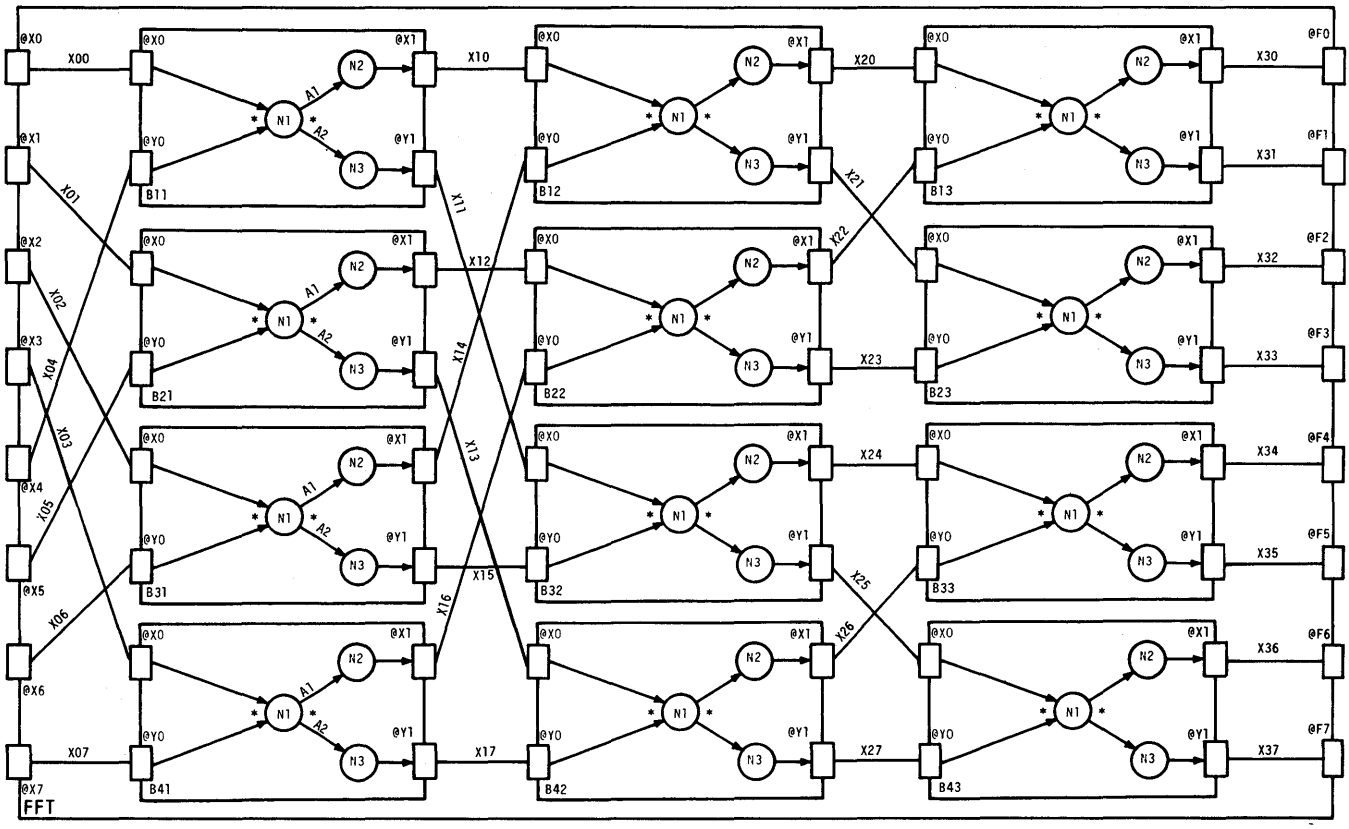


Figure 8—Control graph for 12 butterfly FFT

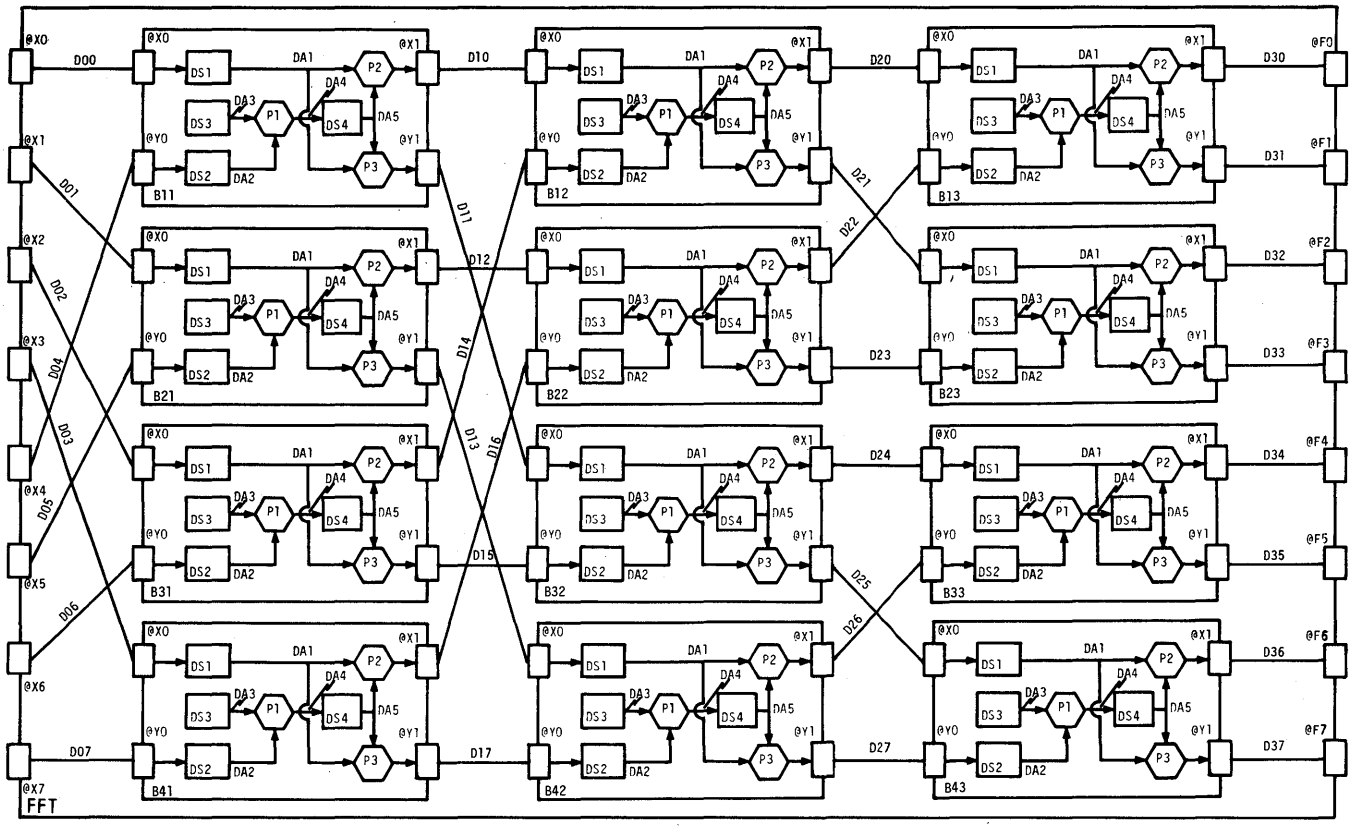


Figure 9—Data graph for 12 butterfly FFT

implemented. Each step was executed independently with existing tools because it uses the same procedure as would be used when creating an original model. The final result for the FFT model is shown in Figures 8 and 9 which display copies of the building block model interconnected to create a concurrent system implementation of FFT. A flow chart of the composition procedure is given in Figure 10.

Let us now consider the evaluation process. The flow-of-control is properly terminating. It can easily be abstracted back into a single control node because it is acyclic and its internal control logic is all "AND."

The data graph interpretation (PLIP) permits us to test and evaluate the FFT system interactively through the GMB simulator. The complete interpreted model was successfully executed. We could enlarge the design to cover a full signal processing system using the FFT as a building block so long as we were consistent with its ENV.

What may be more important is that, in the same manner as we have shown for the MESSAGE TRANSMITTER example, PLIP can be reduced to PL1 code and we have completed a path from programming-in-the-large to programming-in-the-small.

Obviously this is a demonstration problem which will get enriched as SARA matures. We do not presently propose, for each butterfly building block, a machine which can compile and execute PL1 programs. Such a machine may not be available on an LSI chip for a little while. In fact a principal goal of current SARA development is to provide interpretation of the data graph in languages which transform directly to programs that execute on microcomputer building blocks residing in a SARA system architecture environment.

CONCLUSIONS

As a superstructure to be imposed upon the final code or software systems, the structural and behavioral models have the following objectives:

- To show the potential flow of resource names between the structural components of a software system.
- To give a designer complete control over the distribution of information between modules. Sockets are the place where one states one's intent with regards to identifier visibility and access rights among modules at any level. The sharing of resource names is facilitated, without unwanted spread of accessibility, that is, different types of access are distinguishable (read-only, read-write), and access to different subsets of a set of resources can be specified. The idea here is to enforce the specifications over the source code at compile/link time.

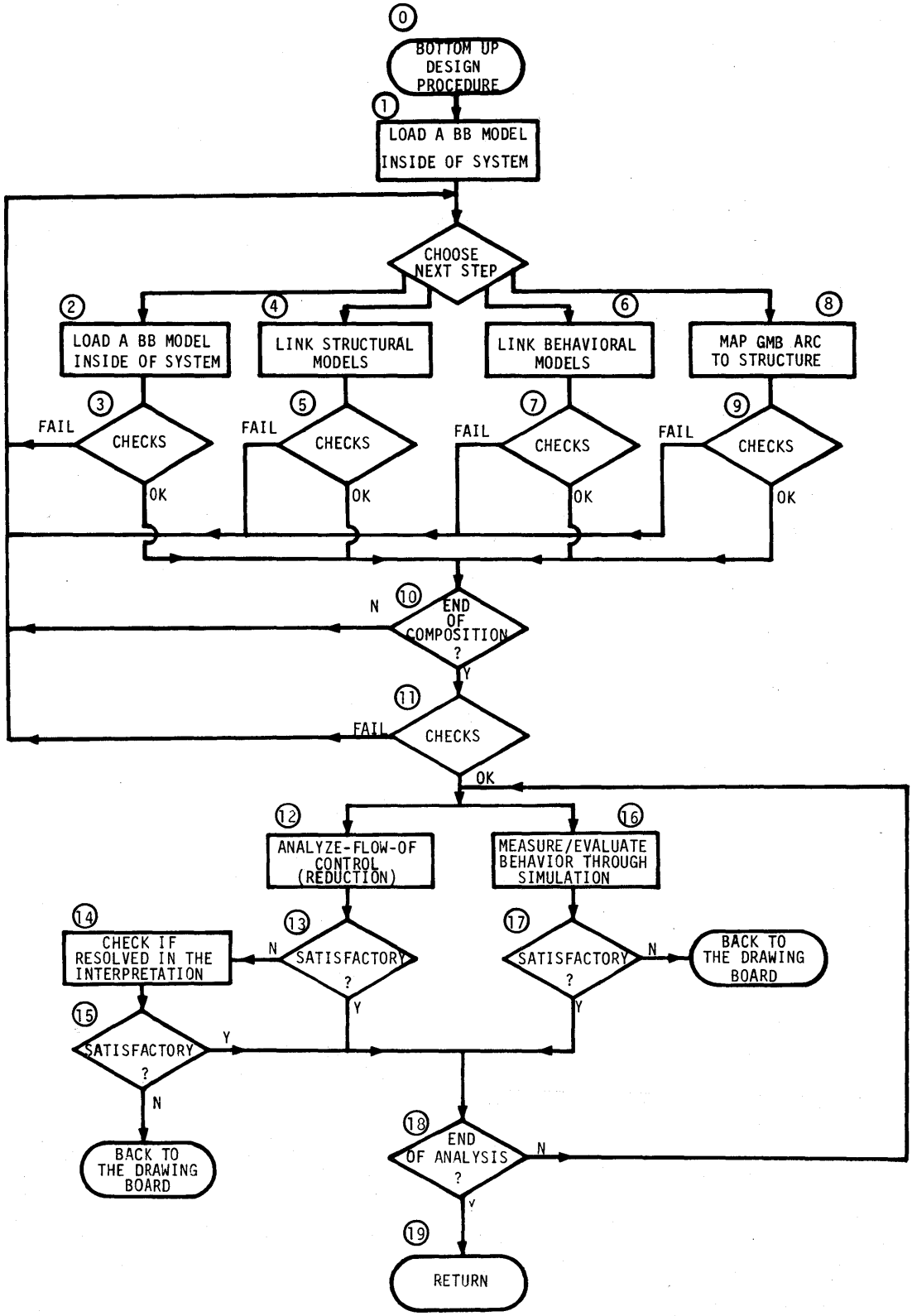


Figure 10—Composition procedure flow chart

- Both top-down and bottom-up design strategies are supported by multilevel modeling facilities.
- Hierarchical organization is enforced, and the structural model, in conjunction with the flow-of-control and flow-of-data models, can keep around conceptual entities that would otherwise vanish in the design process. The whole design tree is kept, not just the leaves, thus making the final product comprehensible both top-down and bottom-up. The SL1 model and the list of attributes associated with each of its sockets encompass what has been proposed in the literature[5,34] as a MIL (Module Interconnection Language). Our approach is more general in that we can define and enforce interface commitments among concurrent software systems.
- To increase the validity of the final product, all mappings and checks for completeness and consistency can be done with machine assistance through SARA.

With respect to flow-of-control, for example, SARA provides analysis facilities to check for proper-termination. A properly terminating flow-of-control model is reentrant, deadlock-free and has determinacy, i.e., regardless of the relative order of execution of the subprocesses involved, the unique output arc is reached and, aside from the vacated entry arc, the state of the control graph is the same as it was initiated to.

The main contribution of the SARA system for software synthesis is the fact that it offers a continuous path from programming-in-the-large all the way down to programming-in-the-small of possibly concurrent systems, while providing a designer with interactive tools and checking procedures to enforce consistency between requirements, structure, function and behavior.

There are a number of issues which require further study:

- The statement of requirements of a design is the cornerstone of the use of the design methodology. So far the English language description of this set is used in SARA. A more formal scheme for describing requirements is needed, as well as the associated criteria to evaluate whether or not a given design satisfies the previously stated set of requirements.
- The structural and behavioral models cited in this paper (SL1 and GMB) are restricted to representing static topologies. We currently handle any dynamic behavior in the interpretation associated with the data graph (i.e., in PLIP). Many interesting design problems, especially in the area of operating systems could be represented in a richer form if we could model and smoothly interface with a modeling scheme in which both the structure and the topology of the behavioral models could change with time, during execution. Other desirable features would be the ability to model recursion, dynamic storage and process allocation (i.e., everything requiring unbounded amounts of a resource) at the level of the GMB control and data graphs.
- Refinement and abstraction rules for GMB primitives are proposed in Reference 27, but they are fairly restrictive. The difficult problem here is to preserve "equivalence" between the two adjacent levels, both during refinement and abstraction.
- Many interesting asynchronous systems, especially in the process-control area are interrupt-driven. The modeling of interrupts in a GMB generates at least two control arcs per interrupt, which tends to overcrowd the control graph. Some extension to the current workings of the token machine should allow for a simpler way of modeling perhaps a restricted class of interrupt handling schemes, such as only allowing one interrupt handler for all interrupts.
- Currently, the only way two GMB processors can transfer information between them is through the use of datasets, which act as external (global) variables for the processors involved. We would like to provide designers with a means of passing parameters between GMB processors (although we might implement them internally in SARA as global variables) in order to make the process of translation from model to code easier.

REFERENCES

1. Dijkstra, E. W., "The Humble Programmer," 1972 ACM Turing Award Lecture, *CACM* 15: 10, October 1972.
2. Parnas, D. L., "On the Criteria to be Used for Decomposing Systems into Modules," *CACM*, 15: 12, December 1972, pp. 1053-1058.
3. Yourdon, E. and L. L. Constantine, *Structured Design*, Yourdon Inc., New York 1976.
4. Parnas, D. L., "The Influence of Software Structure on Reliability," *SIGPLAN Notices* 10:6, June 1975.
5. Thomas, J. W., "Module Interconnection in Programming Systems Supporting Abstraction," Computer Science Program Technical Report No. CS-16, Brown University, Providence, R.I., April 1976.
6. Wirth, N., "Program Development by Stepwise Refinement," *CACM* 14:4, April 1971, pp. 221-227.
7. McGowan, C. L. and J. R. Kelly, *Top-Down Structured Programming Techniques*, Petrocelli/Charter, New York, N.Y. 1975.
8. Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming*, Academic Press, London, 1972.
9. Dennis, J. E., "Modularity," in F. L. Bauer (ed.), *Advanced Course on Software Engineering*, Springer-Verlag, New York, 1973.
10. Parnas, D. L., "A Technique for Software Module Specifications with Examples," *CACM* 15:5, May 1972, pp. 330-336.
11. Liskov, B. H. and S. N. Zilles, "Programming with Abstract Data Types," *Proceedings of a Symposium on Very High Level Languages*, in *SIGPLAN Notices*, 9:4, April 1974, pp. 50-59.
12. Popek, G., J. J. Horning, B. W. Lampson, R. London, and J. Mitchell, "Notes on the Design of Euclid," *Proceedings of the ACM Conference on Language Design for Reliable Software*, Raleigh, North Carolina, March 28-30, 1978.
13. Wulf, W. A., "ALPHARD: Toward a Language to Support Structured Programs," Technical Report, Carnegie-Mellon University, April 1974.
14. Crocker, Stephen D., "State Deltas: A Formalism for Representing Segments of Computation," Ph.D. in Computer Science, University of California, Los Angeles, August 1977.
15. Estrin, G. and R. Turn, "Automatic Assignment of Computations in a Variable Structure Computer System," *IEEE Transactions on Electronic Computers*, Vol. EC12, No. 5, December 1963.
16. Martin, D. F. and G. Estrin, "Experiments on Models of Computations and Systems," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 1, Feb. 1967, pp. 59-69.
17. Gostelow, K. P., V. G. Cerf, G. Estrin and S. Volansky, "Proper Termination of Flow-of-Control in Programs Involving Concurrent Processes," *Proceedings of the ACM Annual Conference*, Boston, August 1972, pp. 742-754.

18. Cerf, V. G., "Multiprocessors, Semaphores, and a Graph Model of Computation," Report UCLA-ENG-7223, Computer Science Department, UCLA, April 1972.
19. Wingfield, M. A., "The Design of an Extensible Processor," Ph.D. in Engineering, University of California, Los Angeles, CA., June 1970.
20. Postel, J. B., "A Graph Model Analysis of Computer Communications Protocols," Ph.D. Dissertation in Computer Science, UCLA, 1974.
21. Yavne, M., "Synthesis of Properly Terminating Graphs," Report UCLA-ENG-7434, Computer Science Department, UCLA, May 1974.
22. Potash, H. A., Tyrill, D. Allen, S. Joseph and G. Estrin, "DCDS digital simulating system," *AFIPS Conference Proceedings,* Vol. 35, Fall Joint Computer Conference, 1969.
23. Estrin, G., R. R. Muntz and R. Uzgalis, "Modeling, measurement and Computer Power," *AFIPS Conference Proceedings,* Vol. 40, pp. 725-738, Spring Joint Computer Conference, 1972.
24. Estrin, Gerald, "A Methodology for Design of Digital Systems—Supported by SARA at the Age of One," *AFIPS Conference Proceedings,* Vol. 47, 1978 National Computer Conference.
25. Gardner, Jr., R. I., "A Methodology for Digital System Design Based on Structural and Functional Modelling," Report UCLA-ENG-7488, University of California, Los Angeles, CA., January 1975.
26. Fenchel, R. S., "System ARchitect's Apprentice (SARA) System Reference Manual," Computer Science Department, University of California, Los Angeles, May 1977.
27. Campos, I. M., "Multilevel Modeling for Synthesis of Reliable Concurrent Software Systems," Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, 1977.
28. *Proceedings of Symposium on Design Automation and Microprocessors,* Palo Alto, CA., Feb. 24-25, 1977.
29. Overman, W. T., "Formal Verification of GMBs," Internal Memorandum, Computer Science Department, University of California, Los Angeles, July 1977.
30. Cooley, J. W. and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Math. of Comput., Vol. 19, April 1965, pp. 297-301.
31. Brinch-Hansen, P., *Operating Systems Principles,* Prentice-Hall, Englewood Cliffs, N.J., 1973.
32. Brinch-Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering* SE1:2, June 1975, pp. 199-206.
33. Modell, H. S., R. G. Ward and T. M. Sparr, "Coordination of Parallel Processes in PL1," Technical Report, Computer Science Department, University of Texas, Arlington, Texas, 76019.
34. DeRemer, F. and H. H. Kron, "Programming-in-the-Large versus Programming-in-the-Small," *Proceedings of International Conference on Reliable Software in SIGPLAN Notices* 10:6, June 1975, pp. 114-121.

GM network station—A low cost graphics system for body tooling

by THOMAS J. RENO
General Motors Corporation
Warren, Michigan

INTRODUCTION

Over the years, General Motors has been actively involved in both the development and use of Computer Aided Design/Computer Aided Manufacturing systems to aid in the design, engineering and manufacturing of automobiles. The use of CAD/CAM technology permeates most of our operations, but nowhere is it as important and pervasive as in the body tooling process. The demands for safety, environmental protection and energy conservation have required a responsiveness that only CAD/CAM systems could provide. No longer can we afford the long lead times of 3+ years previously required to go from design concept to steel. Today we must respond quickly and effectively to an ever changing car market—and CAD/CAM is helping us to do so by dramatically shortening the body tooling cycle.

The impact of CAD/CAM on our body tooling process has been demonstrated many times, but never more convincingly than in the years following the 1974 oil embargo. This new technology enabled us to produce the 1976 subcompact Chevette in only two years in response to a then rapid shift of public sentiment to small cars.

CAD/CAM contributed heavily to the design and tooling of our 1977 full size cars. In fact, Mr. Howard Kehrl, GM Executive Vice-President, stated in a Business Week article that only through use of this technology were we able to produce our 1977 'B' and 'C' models *at all,* after a nine month initial delay and a major shift in specifications halfway through the program. Computer graphics was also used to evaluate the structural characteristics of proposed designs and to eliminate much of the trial-and-error process previously involved in making design changes.

CAD/CAM has truly come of age and is now an integral part of our business. It is difficult to imagine how we could design and build a car without it. But CAD/CAM did not just happen in General Motors. Its acceptance today is the culmination of a long process begun in the late 1950's when the possibilities of the newly emerging technologies of Numerical Control and visual man-computer interaction were first recognized.

SYSTEMS ORIGINS

The GM Research Laboratories pioneered man-computer communication through their DAC (Design Augmented by Computer) project. At the same time, GM Manufacturing Development was developing N/C technology for sculptured body tooling. Significant resources of both people and money were devoted to these projects and to related developments in the areas of mathematics and computer science until usable prototype systems were available. These prototype systems included sophisticated new mathematics capable of representing the types of curves and sculptured surfaces found on automobiles; data bases and data structures capable of storing these mathematical models; regional milling capabilities for both ball cutters and end-mill cutters; and user interfaces which permitted the easy addition of other new applications such as structural analysis, vision studies, etc.

In 1967, Manufacturing Development was assigned the responsibility of transforming these prototype systems into usable production tools. The result was two complementary systems, CADANCE (Computer Aided Design and Numerical Control Effort) and INCA (Integrated Numerical Control Approach), which together provided an effective CAD/CAM base for design, engineering and tooling. Additional application and system development was undertaken by Fisher Body and Design Staff to finally bring all of the development efforts into full production use. The systems in use today at GM are the result of all these combined efforts.

COMPUTER AIDED DESIGN BACKGROUND

The CAD systems in General Motors were all developed as outgrowths of the original DAC system and used computer graphics and time sharing as the base upon which they were built. Today approximately 75 graphic consoles served by two large computer installations at the GM Technical Center are employed by our divisions and staffs to do body

design and engineering work. The CAD systems all employ refresh graphic display devices and accept virtually all of their input via light pen selection from the face of the screen. Similarly, all of the output is transmitted to the user via displays on the screen. This combination of time-sharing, graphic input and graphic output provides an unbeatable combination for problem solving. Mistakes are detected instantly and the problem can be re-run at once. Long delays waiting in input and output queues are totally eliminated. The lead time savings achieved using this technology are truly impressive. Very often 6-8 weeks are eliminated from a conventional 12 week job.

COMPUTER AIDED MANUFACTURING BACKGROUND

The CAM systems developed for body tooling exploited the technology of Numerical Control and used the computer facilities available at the time. This was, as mentioned earlier, in the late 1950's and 1960's. At that time, batch mode computer runs using card input and obtaining printed output was the standard bill of fare. Some minor improvements such as Remote Job Entry (RJE) facilities and Time Sharing Option (TSO) were added to the batch systems, but the basic philosophy of our CAM programs remained unchanged. There was no way to avoid the long waits in input queues and output queues, the randomness of courier deliveries, or the large stack of printed output which had to be examined for possible errors.

The CAM programs, such as INCA and APT, used by GM were developed to process the masses of numerical control data required for wood model and die construction. Human decisions were emulated within these programs wherever feasible or possible. Nevertheless, human intervention was required to analyze and evaluate intermediate results produced by these computerized processing aids. These results were in the form of computer listings and usually had to be converted to graphical form via sketches or drafting machine drawings. This process obviously was very time consuming.

For a number of years, this mode of operation was accepted, although the long time delays associated with each step were excessive. The technology of N/C, however, progressed and saved time and money in spite of long processing times. Indeed, instead of working to improve the process, we worked on improving the technology of N/C. Regional milling techniques and flow line cutting were honed to perfection. Sophisticated evaluation techniques and surface linking programs were developed and brought into production use. A data storage and retrieval program which virtually eliminated data cards was perfected and has been operational since the late 1960's. CAM technology indeed grew to where today nearly all full size interior and exterior sheet metal wood models are cut using N/C.

REASONS FOR CHANGE

All the while, however, the operational procedures remained unchanged. The CAM systems became a victim of the computer technology upon which they were based. The batch mode operating systems did not allow any meaningful operational improvements to be made. It still took several weeks to produce a complete set of cutter tapes for a complex wood model. The CAD systems, on the other hand, had benefited from new advances in computer hardware and software and were achieving remarkable results in both processing capability and lead time reduction. The N/C lead times began to be the bottleneck and it soon became apparent that improvement in processing techniques was essential.

Based on the effectiveness that computer graphics had demonstrated in the area of Computer Aided Design, it was obvious that this technology could provide similar processing improvements to our Computer Aided Manufacturing systems. However, it was not obvious exactly how to proceed to implement a graphics CAM system in GM. Since we were attempting to modernize an existing operation—not start a new one—we had to take into account constraints the existing program imposed on us. These were geography, cost, and time.

GEOGRAPHICAL CONSTRAINTS

In GM, the vast majority of design activities occurs within the square mile area of the GM Technical Center. Tool manufacturing, however, encompasses a much larger area and even extends to West Germany where Adam Opel is an active participant in our CAM activities. Any modification of our CAM system for body tooling had to encompass *all* tool manufacturing. Computer graphics—even today—has difficulties economically locating consoles more than one mile from the computer. Consequently, any move of our CAM systems towards computer graphics had to be independent of the large Technical Center computers.

COST CONSIDERATIONS

As to cost, the batch N/C systems had been tuned over the years to the point where processing charges were quite minimal. Also, CAD via graphics had created an excellent data base from which our CAM activities could begin. This meant that very often the full power of the large Technical Center computers was not needed for our CAM work, and if we used them we would be incurring greater processing costs than necessary. This again indicated that a solution independent of the large Technical Center computers was required.

IMPLEMENTATION TIME CONSIDERATIONS

The repertoire of computer programs used in die model construction is very extensive. All of the programs are required in some degree to complete the job. Any modernization of our CAM system would have to take this into account and provide access to similar facilities. Since it would be impractical and very expensive to convert all ex-

isting programs to a graphical format, any new system would have to be compatible with the in-place system.

SYSTEM SPECIFICATION

With these considerations in mind, Manufacturing Development began exploring hardware/software combinations that would meet our objectives. We needed a system that was inexpensive, could operate independently of large computers, and yet could communicate freely with the existing in-place systems. In 1973, we began development of a *Low Cost, Stand-Alone N/C Review System* to bridge the gap between CAD and CAM. This system answered the questions of geography, cost and time and seemed to be a start towards getting the benefits of graphics into our body tooling manufacturing.

HARDWARE COMPONENTS

The system developed consisted of:

Mini-computer with Disk Storage
Interactive Graphic Terminal
Medium Speed Dial Phone Line
N/C Tape Punch

Its total purchase price is approximately $75,000, which equates to less than $10/hour for a one-shift operation. This is well below the cost of large central time-sharing computers and even less than batch system costs. The system is indeed a stand-alone system. It can be placed in any room of any plant and requires no special wiring, floors, or unusual cooling. Its dial-up phone line and N/C tape punch enable remote units to communicate with each other or with the central computers at the Technical Center. These features overcome the geographic and implementation time constraints imposed on our CAM system.

HISTORY AND INITIAL IMPACT

Initially, this system simply provided a graphic input and output facility for our existing batch mode CAM programs. Substantial lead time was gained by simply reviewing data and input commands graphically before submitting them, thereby detecting errors early in the game and correcting them *before* batch runs rather than after. Similarly, graphic review of the output was faster than laboriously scanning pages of printout or waiting for full size drawings to be made on already overloaded drafting machines.

The initial components of the system were few. There was a data base manager to store and retrieve information. Function buttons were available to change views, shift, scale, rotate data, etc.

In addition to these graphic data review features, other computational programs were available to:

Create arcs, circles, straight lines
Modify Data (delete, insert, move points)
Transform Data (Rotate, Translate, etc.)
Create Card Images for Job Submission to Large Computers
Teleprocess
Punch Mylar Tape

This simple review system allowed us to export Computer Aided Manufacturing technology to our plants and was a start towards bringing the benefits of computer graphics to the manufacturing floor.

A CHANGE IN DIRECTION

The use of this low-cost review system never became very pronounced, however, because even while doing our initial development work on the system we realized the potential we had at our fingertips. It became readily apparent that the mini-computer was never really being taxed. The users also posed the question, "Why, since I have all my data here, can't I generate my N/C tapes here?" This, then, marked the turning point of the project and brought about the renaming of the system to the 'Network Station.' This name symbolizes two things. First, the system is still part of our overall CAD/CAM program and depends on access to the central data bases via the telephone network. Second, once the data is received, the system operates as an independent station until the job is complete. The term 'Network Station' seems to fit nicely the function and use of the system.

DEVELOPMENT AND CURRENT CAPABILITIES

User divisions enthusiastically accepted this change of direction and even contributed manpower to assist in the software development. Eight departments representing five divisions and staffs have participated in the Network Station development. Some of the items added to the software repertoire to make possible a full stand-alone operation include:

Additional Curve Generation Functions (Fillets, Sweeps, etc.)
Curve Smoothing
3-D Meshing
Surface Development
Surface Linking and Trimming
N/C Machining
Interactive Cutter Location Verification and Modification

Through the use of this system, it is now possible to completely process our most difficult wood models without using any of our old batch CAM programs. This includes surface definition, cutter location, and N/C tape punching. Divisions with easy access to the Technical Center computers still find it more convenient to use the existing old CAD/CAM

interfaces, but they are all acquiring Network Stations for future use. Someday this new graphic CAM system may make our old batch systems completely obsolete.

NETWORK STATION DATA FLOW

The Network Station offers a range of CAM uses never before available in our body tooling process. The ability to teleprocess information, read in punched tapes, or create data directly on the system provides for a wide variety of different input sources. For example, inputs from all of these sources are a commonplace occurrence today.

Teleprocessed Information from CAD Data Bases
Digitized Information on Punched Tape
3-D Input from Hard Model Scanners
Cutter Location Information on Punched Tape
Experimental Drawings

These input sources cover all of the common ways in which data is now collected. Once entered into the system, the user can proceed to process the model. When finished, a number of choices are available for outputting results. These include:

Punched Tape for Computer Controlled Mills
Punched Tape for Drawing Machines
Data for Teleprocessing
(a) To a Large Computer for Additional Processing
(b) To a Remote Site for Machining, Viewing, etc.
Printed Process Sheets
Printed Checking Information

NETWORK STATION ACCOMPLISHMENTS

As a testament to the productivity and flexibility of the Network Station, Fisher Body, Die Engineering Activity, now uses this system for virtually all of their CAM work related to body tooling. They have six systems which they now use to produce:

All Wood Die Models (Inners & Outers)
All Seat Bun Patterns
All Bend and Sweep Gages
Production Glass Block Tapes
Die Profile Tapes

DIVISIONAL INSTALLATION

Fisher Body is by far the largest single user of our Network Station. They are not, however, the only user. Seven divisions and staffs have equipment either installed or on order. Several others are actively pursuing authorization for equipment. The tally as of now is 18 units installed, 3 awaiting delivery and 3 in the process of approval. The divisions also have tentative plans for 10-12 additional units in the next two years. These units are located as far away as West Germany, thus demonstrating the true stand-alone nature of the system.

GROWTH OF CAD/CAM CENTER

As you can see, the Network Station is rapidly becoming the dominate force in our N/C body tooling system. The growth of this system is well in tune with the expanded use of our Computer Aided Design programs and indeed complements and supplements our design and engineering systems. As the use of CAD grows, it too is spreading out geographically to our outlying divisions and plants. We are even now locating graphic design consoles in Lansing, Pontiac and Flint. The hardware used for both our CAD and CAM systems is interchangeable and thus permits multiple use of the same equipment.

MULTIPLE USES OF EQUIPMENT

For example, a typical remote installation consists of:

Mini-computer and Peripherals
Graphic Console
Drafting Machine

When the mini-computer and the Graphic Console are connected, the result is a Network Station. The Design Console can be connected over high-speed telephone lines to the central CAD computers, thus allowing the mini-computer to be used for teleprocessing, other computing applications, or as an input source for the drafting machine.

The drafting machine, of course, can also operate independently of the mini-computer. This combination of hardware and software allows full use of all equipment, provides access to both CAD and CAM programs, and allows the user to operate either as part of the central design and engineering community or as a stand-alone station. Also, since no specialized hardware is used, it is easily maintained, can be inexpensively upgraded for more power, and a wide range of peripherials such as plotters, digitizers, etc., can be easily added.

ADAPTABILITY OF SOFTWARE

The intent of the Network Station has always been to get some computer power out to the plants where the work is actually done. Realizing that each plant operates somewhat differently, and knowing that one central development activity could never fill all of the needs of all the plants, we designed the Network Station system to be very easy to program. All application programs are written in Fortran. All input, output, and graphic commands are available to the programmer via simple subroutine calls. A simple programmer's manual is available which describes how to write Network Station programs. In addition to our own staff, we have trained twenty-two people representing six divisions

and staffs how to add applications to the system and they have all contributed to some degree. Our experience has been that a programmer knowledgeable in Fortran can be an accomplished Network Station programmer in a period of approximately four weeks.

POTENTIAL FOR COMPUTER AIDED MANUFACTURING

The impact of computer graphics on the GM body tooling cycle has been very substantial. But full lead time and dollar savings will not be achieved until the entire body tooling cycle from design concept to production die is affected. To date the major impact of this new technology has been in the area of product design. Our large graphics systems have been profitably used in body design and body engineering for several years. However, examination of the distribution of work in the body tooling cycle shows that 88 percent of all hours are expended in tool design and construction. Similarly, 70 weeks are consumed in this effort as opposed to only 17 weeks for panel design. Clearly, the greatest potential savings are in these areas, and we have scarcely begun to impact them.

THE BODY TOOLING SCENARIO

What is ultimately required is a general machining-manufacturing system for body tooling. This system would use high performance Numerical Control machines to do the milling and inspection of die models and dies. The processing, communications, and coordination would be done on small computers linked together by telephone lines. The system we envision would consist of at least three major parts.

The first would be Tool Design. Here the die models, dies and fixtures would be designed. The product design data would be used to determine how many and what kinds of dies are needed to produce a part and also the sequence in which the dies would be used. The amount of die tipping would be determined and the runoff conditions developed. Binders and die and checking fixtures would be designed. All the necessary drawings would be output.

The second part of the system would deal with the machining of the die models, dies, and related tools. The product design and tool design data would be used to generate N/C tapes to drive the mills. These tapes would contain all the machining information. They would be verified and used to machine wood die models, gages, and glass blocks. Tapes to cut metal would be produced either from the design data or by tracing the wood die models with computer controlled scanners. All tapes, process sheets, and listings that are necessary for setup and milling would be produced. If the use of computer controlled inspection machines becomes widespread, these tapes would also be generated.

The third part of the system would ensure proper utilization of equipment and metal in the stamping plants. Offal would be scrutinized to see if other parts could be produced from it. The press configurations and material handling equipment would be checked to see if they were available and not overloaded. Once this was assured, layout drawings would be generated and sent to the plants. In addition, material deviation studies would be done to ensure that no plant was using a higher grade metal to produce a part than other plants.

CONCLUSION

We are slowly working towards developing such a system, and the Network Station plays an important role in these future plans. We are already actively working with the divisions to develop programs in the areas of die processing, press planning, clay milling and 5-axis tool center computations. A prototype program for generating N/C inspection tapes is already available on the Network Station. Also, programs to minimize metal waste through optimum blank nesting are now under way. We are convinced that for N/C technology to succeed, it must become the "conventional process." The Network Station is helping this happen by placing the computer in the plant where the work is done and by removing the mystique from Computer Aided Manufacturing.

In summary, we have a tiger by the tail in our Network Station project. He has "devoured" every task given to him and still seems to have a voracious appetite. We are letting go of the tail a little at a time as we assign more and more of the new body tooling applications to him. We feel that it will take quite a while before either his appetite is satisfied, or we "run out of tail."

Computer aided design (CAD)/technical data automation (TDA)

by VERNON R. PEARL
U.S. Army Research and Development Command
Aberdeen Proving Ground, Maryland

INTRODUCTION

The Technical Data Package (TDP) is the terminology used by the U.S. Government to describe the documentation generated during research and development for the procurement of end items.

Spiraling costs in producing TDP's have long been a major concern of the Department of the Army. Thousands of engineering drawings, specifications, standards, and other related technical data that make up a TDP are produced each year. These TDP's are used to document, in detail, the Army materiel end items required to supply troops in the field and to maintain a state of constant military readiness. With industry and Government continuing to change the technological base in both design and manufacture, keeping documentation up to date has become costly and difficult. Because of the continually changing technology, end items may soon become obsolete and many of the repair parts required to maintain fielded items would no longer be manufactured. In either case, it's "back to the drawing board" to start a new design, improve an existing one, or rework items in the field to allow for procurable spare repair parts.

At the CSL-Technical Support Division, ARRADCOM, which services the CSL at Aberdeen Proving Ground, Maryland, there are approximately 50,000 engineering drawings on chemical-related items. There is also a vast number of standards and specifications associated with these chemical items. New chemicals are constantly being developed. Sometimes the specific chemical required to produce a certain munition is not always available. These factors contribute to a high rate of both drawing and specification changes and lead to many contract waivers and deviations. All of these problems result in changes to the technical data and higher documentation costs. It is for these reasons that we begin looking into using the computer in an effort to reduce both time and costs and to streamline the entire technical data generation process.

EXISTING EQUIPMENT AND SOFTWARE

The equipment selected for accomplishing computerized engineering drafting is as follows:

1. A plotting and digitizing system (manufactured by Actron Industries).

This system was purchased in 1972. It consists of a 5- by 12-foot drafting table and a control console containing a digital computer and other electronic equipment related to monitoring and measuring functions. In addition, the console is equipped with a closed-circuit TV monitor providing the operator with a magnified view of the drawing area and enabling him to select individual points for digitizing or to monitor and supervise an automatic drafting operation (Figure 1). A drafting and digitizing head suspended from a gantry and armed with an optical line-following device produces the X-Y coordinate data required for line definition. The system is basically restricted to two-dimensional work but certain perspective, isometric, and three-dimensional views can also be produced. In the digitizing process (the ability to automatically define lines and curves), the measurement system continuously reports line locations to the computer. The computer is programmed to process this data into a tool-path description and output a numerically-controlled tape complete with spindle speeds, feed rates, and other miscellaneous functions. The completed tape can be inputted directly to the machine tool without any further need for processing.*

2. A refresh interactive graphics system (manufactured by Information Displays Incorporated).

This stand-alone system was purchased in 1973. It contains a display console, a display controller, a graphic display processor, a teletype station, a magnetic tape unit, and a disc. The display console consists of a 21-inch cathode-ray tube (CRT), a light pen, and an alphanumeric keyboard. The display controller has vector, character, and circle generators, four types of line structures, and four levels of intensity control. The graphic display processor consists of 32K of core. It has hardware multiply/divide, direct memory access, a priority interrupt module, and an automatic boot

* SME Technical Paper MS73-958, "Application of Automated Drafting and Digitizing at Edgewood Arsenal," by Vernon R. Pearl.

Figure 1—Drawing being digitized

strap loader. The magnetic tape unit uses 7-track, 800-bpi dual-density tape. The disc has a 1.1 million word capacity. The drafting and design software consists of drawing lines, circles, arcs, and rectangles. Texts and symbols can also be generated. Other functions include move, plumb, level, copy, delete, and group. The system also contains library support for storing symbols and procedures for saving, starting, getting, and deleting drawings. The system is a general-purpose operating system programmed in Fortran. In 1975, an additional display console was added to the basic system including a time-share capability which allows for simultaneous and independent utilization of either console (Figure 2). In 1976, a third stand-alone system was purchased and is currently being interfaced with the laboratory's Univac 1108 computer. Other software improvements recently added include three-dimensional modeling/dynamic simulation and printed circuit-board-layout routines.

These three systems in conjunction with the drafting/dig-

Figure 2—Two-station interactive graphics system in operation

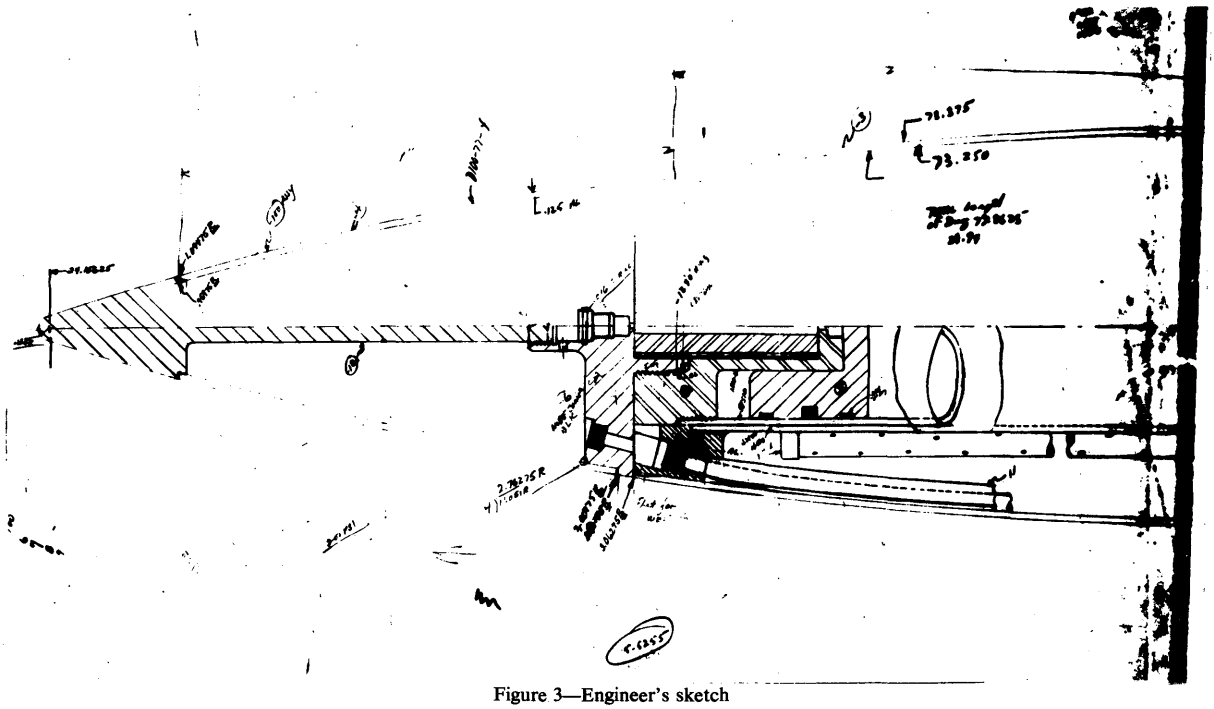


Figure 3—Engineer's sketch

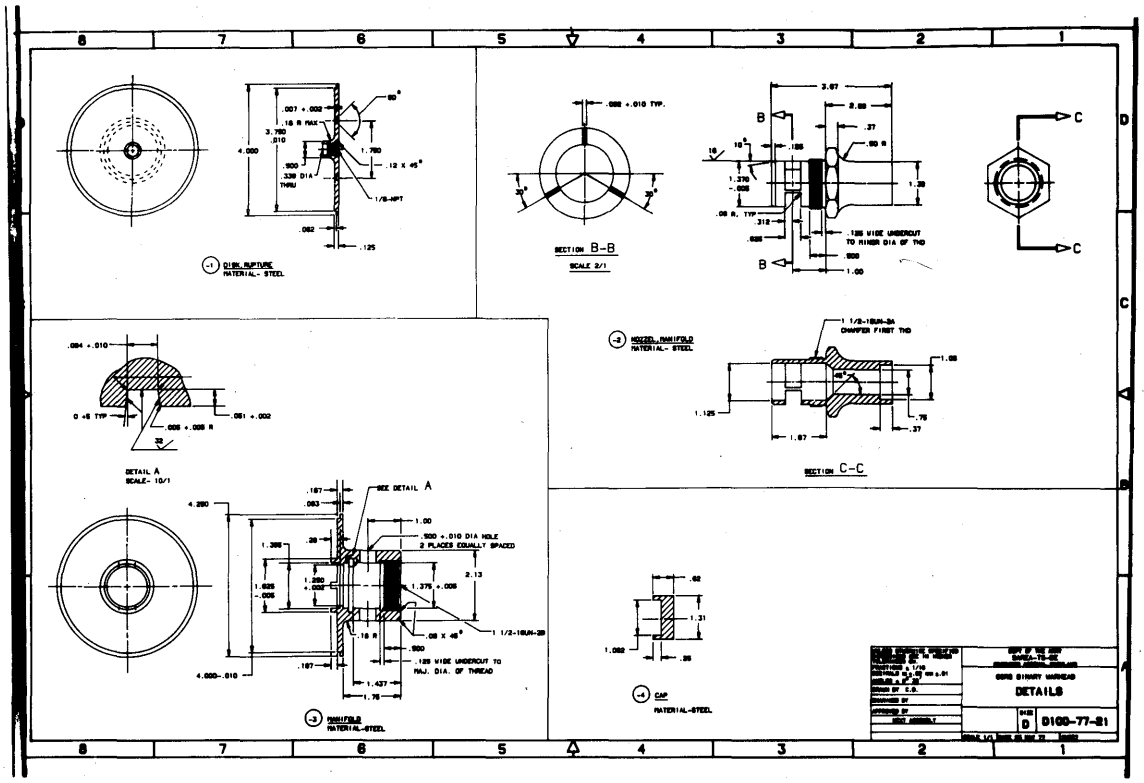


Figure 4—Drawing details

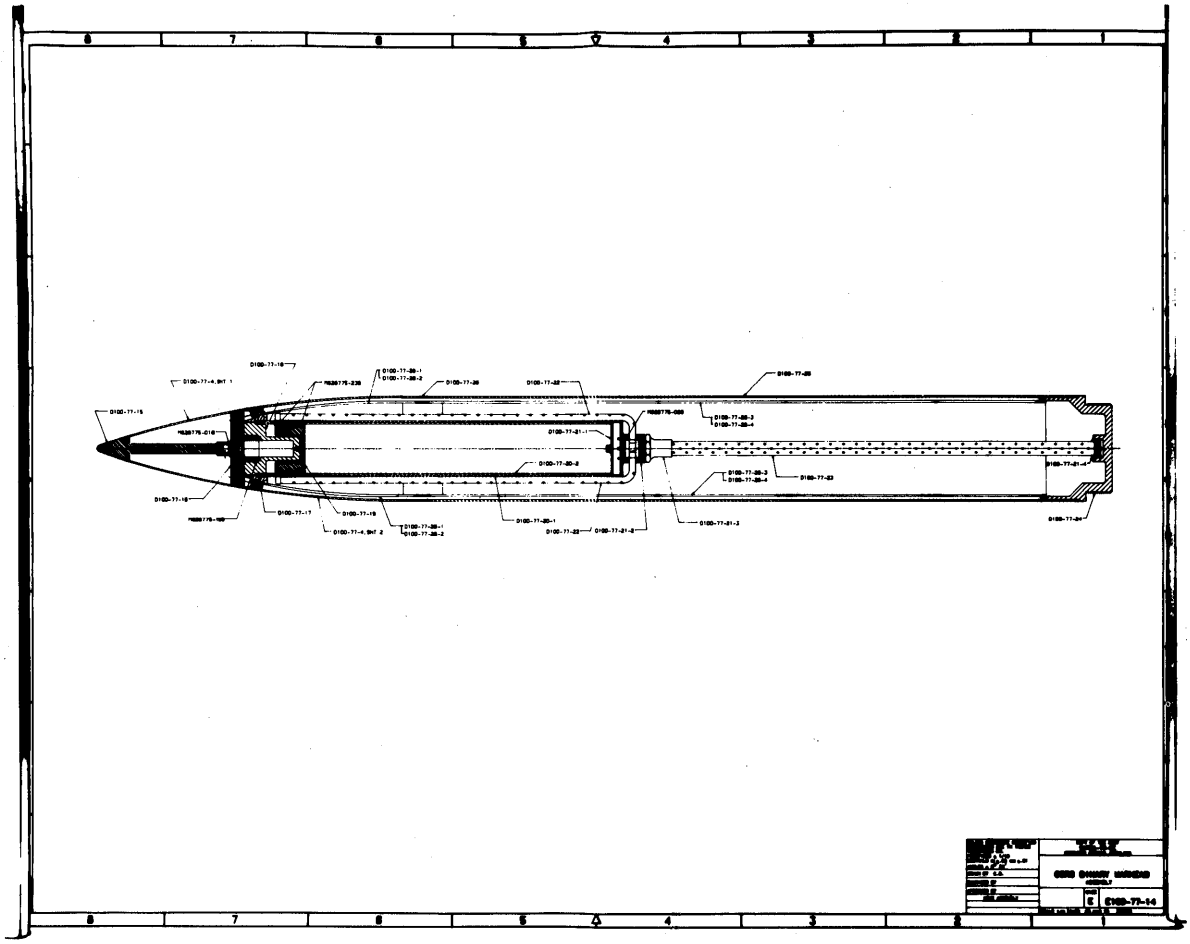


Figure 5—Assembly drawing

itizing system have given us an extremely powerful computer-aided-design system.*

COST-SAVING EXAMPLES

There are many varied cost-saving projects that have been accomplished through the use of this computer-aided-design system. The examples chosen were picked mainly because of their relation to technical data work. Keep in mind, however, that printed circuit boards, numerical control, floor plan layouts, etc., can also be accomplished using CAD.

1st example

This example is one of the more common types performed with the system. An engineer brought in a sketch of a binary munition (Figure 3). He had semi-scaled the sketch to get the proper proportions. With this information we were able to digitize the various proponents using the Actron equipment and ascertain such data as volume, first moment about the Y axis, center of mass along the X axis, second moment about the X-Y axis, radius of gyration, polar moments, and surface areas. The significance of using this program is that while you are designing you can easily determine moments at any time without using any mathematical formula or doing any computer programming. A density factor can be input through the teletype to allow calculations to be made of an object or shape which is constructed of nonhomogeneous material in seconds. The design can be immediately adjusted to achieve the configuration necessary for proper flight stability. Once the engineer is satisfied with the design configuration, the parts are automatically drawn up on the CRT (Figure 4). The parts are then combined to make an assembly drawing (Figure 5). This combining operation checks for both alignment and proper fits, such as does the O-ring engage properly to its mating part. The system also has the ability to automatically dimension any line or circle that has been put on the screen. This method of design allows the operator to make changes rapidly, catch errors, save on fitting calculations, and merge drawings.

Twenty prototype drawings were required to describe the items sufficiently enough for fabrication by the experimental metal shops. By using the computer-aided-design system on

* SME Technical Paper MS77-764, "Cost Effectiveness of Computer-Aided Design (CAD)," by Vernon R. Pearl.

this job, all calculations and drawings were completed within two weeks. It is estimated that approximately two months would have been required if it had been done manually.

2nd example

This area of cost savings comes from the use of a three-dimensional software package with dynamic rotation that resides within the graphic system. The Research Division of the Chemical Systems Laboratory uses this software routine to visualize and verify molecular binding. Before this routine can be used, certain coordinate data describing the molecule must be fed into a molecular coordinate calculation program that resides within the Laboratory's Univac 1108 computer. This program gives the scientist an idea of the shape of the molecule and produces a printout yielding the X, Y, Z coordinates of the various elements associated with it. This data is then fed into a molecular orbital calculation program which produces X, Y, Z coordinates for a minimum energy configuration of the molecule. The scientist has no way to visualize these several sets of data in space. Incorrect data fed to the computer would go undetected. Erroneous data used in further research proved costly in both time and manpower.

To eliminate this problem, the initial molecular coordinate data is now displayed on the graphics system after the initial Univac 1108 data is received (Figure 6). Here it is possible to rotate the entire molecule at any atomic point and at any angle to check the bonding for errors. This verified data is then fed back into the Univac 1108's molecular orbital calculation program. The new data is again displayed on the screen. This becomes quite significant, since the scientist can now see the minimum energy configuration which may bear little resemblance to the original configuration. This visual display allows for a more direct and rational evalua-

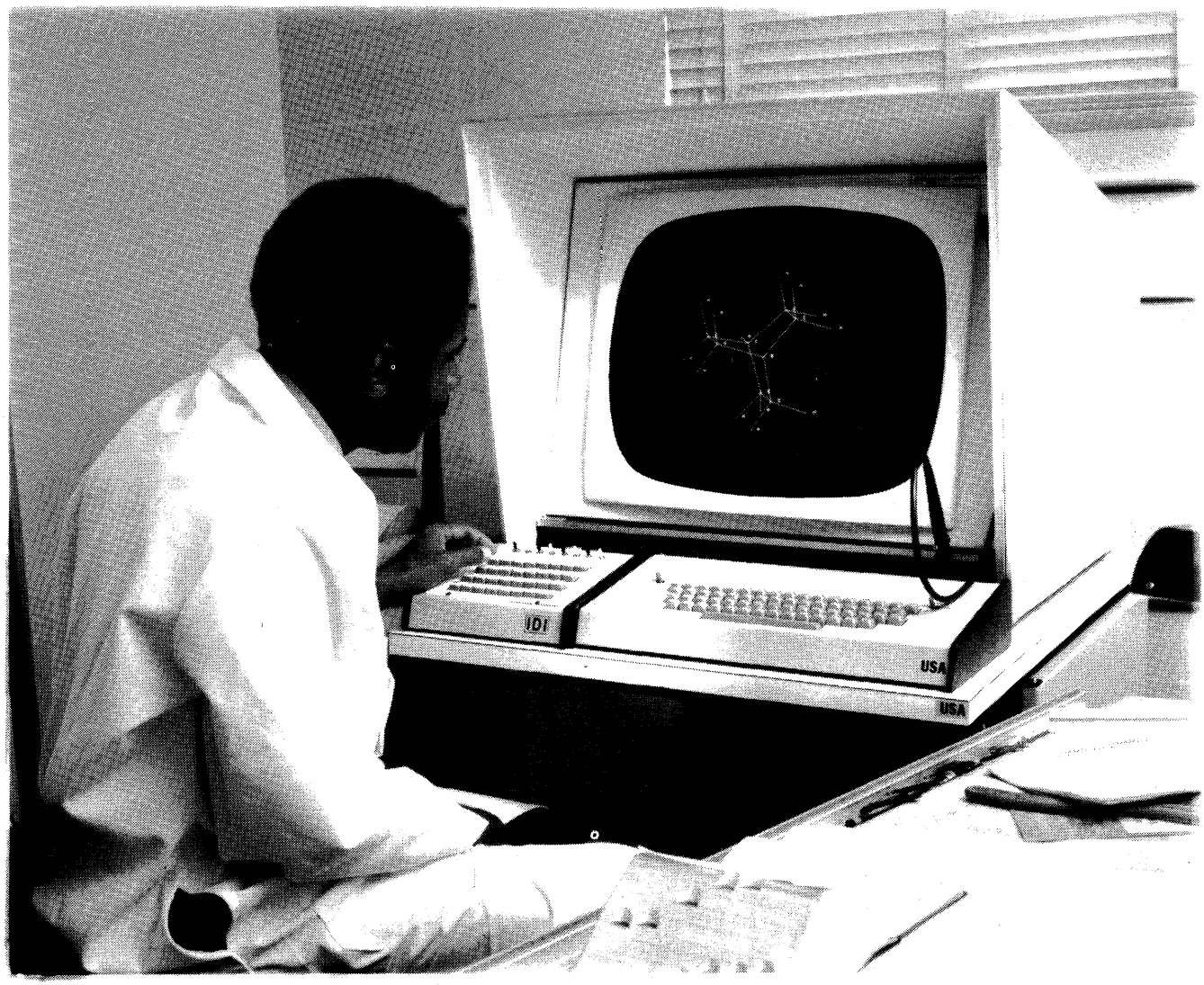


Figure 6—Molecular rotation displayed on the cathode-ray tube

tion and replaces expensive manual simulation methods. We also have the most probable molecular configuration which will subsequently be used in various biological and physical studies. Though the savings are somewhat intangible, one can easily see that by using the computer-aided-design for just one hour per molecular validation many man-hours of the researchers labor are continually being saved.

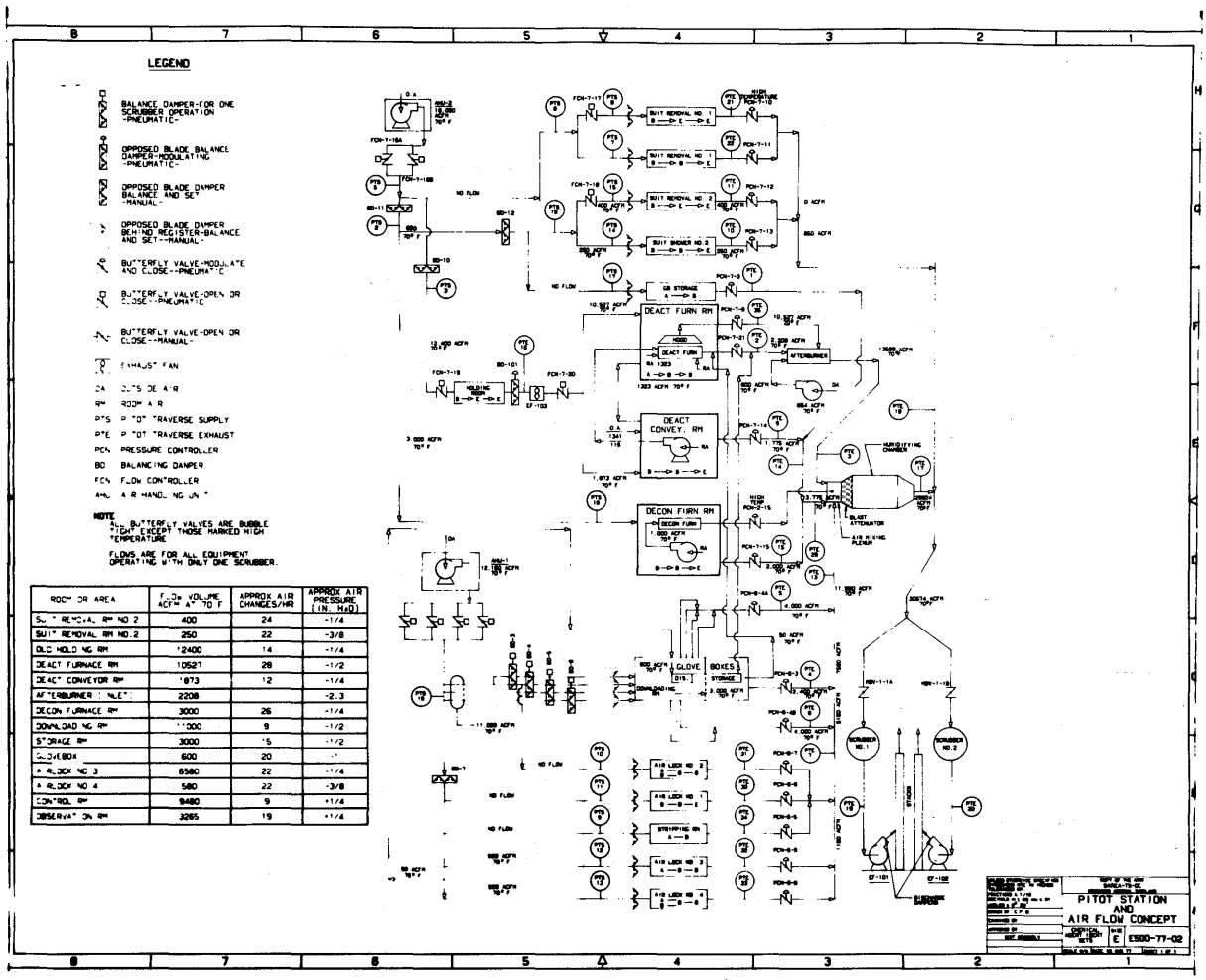
3rd example

This example is a mechanical schematic, which was accomplished with ease by using the graphic system. A freehand sketch giving an idea of the flow was the only requirement needed to get started. Symbols generated months ago and stored were pulled from the library and displayed on the screen. New symbols were generated as required. The plumb and level functions were used to line up these symbols. A line connecting function was used to tie the symbols together. The alphanumeric keyboard was used for labeling. Once again changes were completed rapidly. Sections could be grouped and relocated anywhere in the drawing in seconds (Figure 7).

Including changes, the drawing was completed in one day, and it is estimated that three days would have been required if it had been done manually. Electrical schematics and wiring diagrams are also easily handled using the same techniques.

4th example

The final example depicts a set of production drawings used in a TDP for manufacture by contract. To prepare drawings for a TDP was the primary reason the equipment was purchased. The development of a TDP can take anywhere from two to six years and requires many man-hours of labor and various prototype hardware building and testing methods. Engineering drawings, and changes to them, are involved in nearly every step of the development phase. The cost of producing drawings while going through this cycle,



ROOM OR AREA	FLOW VOLUME ACFM AT 70 F	APPROX AIR CHANGES/HR	APPROX AIR PRESSURE (IN. H2O)
SUIT REMOVAL RM NO 2	400	24	-1/4
SUIT REMOVAL RM NO.2	250	22	-3/8
DEC HOLD NG RM	2400	14	-1/4
DEACT FURNACE RM	10527	28	-1/2
DEACT CONVEYOR RM	873	12	-1/4
AFTERBURNER VALVE	2208		-2.3
DECON FURNACE RM	3000	26	-1/4
DOWNLOAD NG RM	1000	9	-1/2
STORAGE RM	3000	15	-1/2
GLOVEBOX	600	20	-
AIRLOCK NO 3	6580	22	-1/4
AIRLOCK NO 4	580	22	-3/8
CONTROL RM	9480	9	+1/4
OBSERVATION RM	3265	19	+1/4

Figure 7—Mechanical schematic

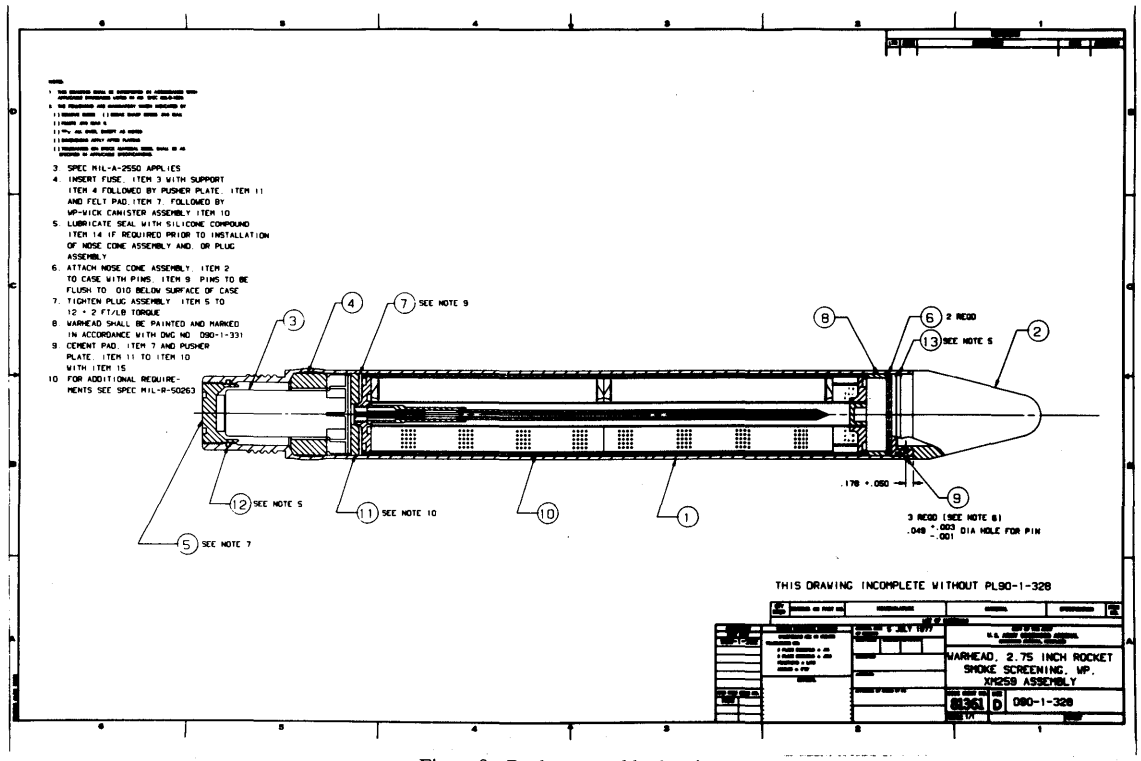


Figure 8—Rocket assembly drawing

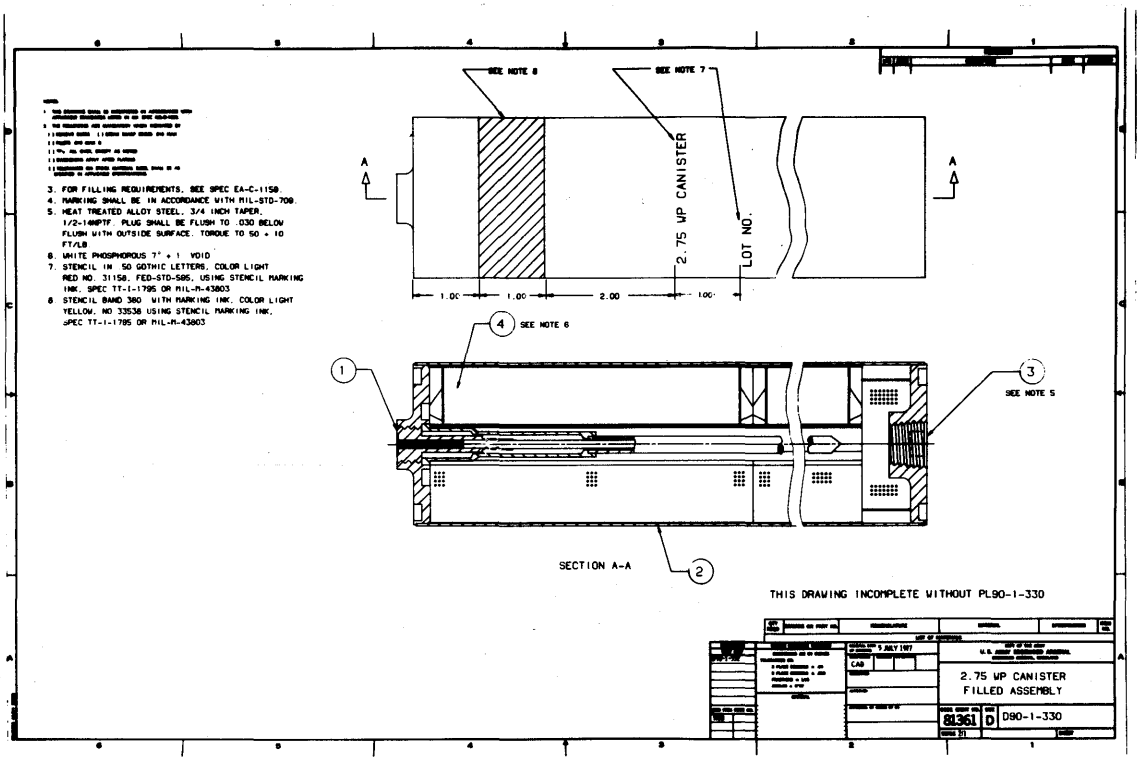


Figure 9—Rocket sub-assembly drawing

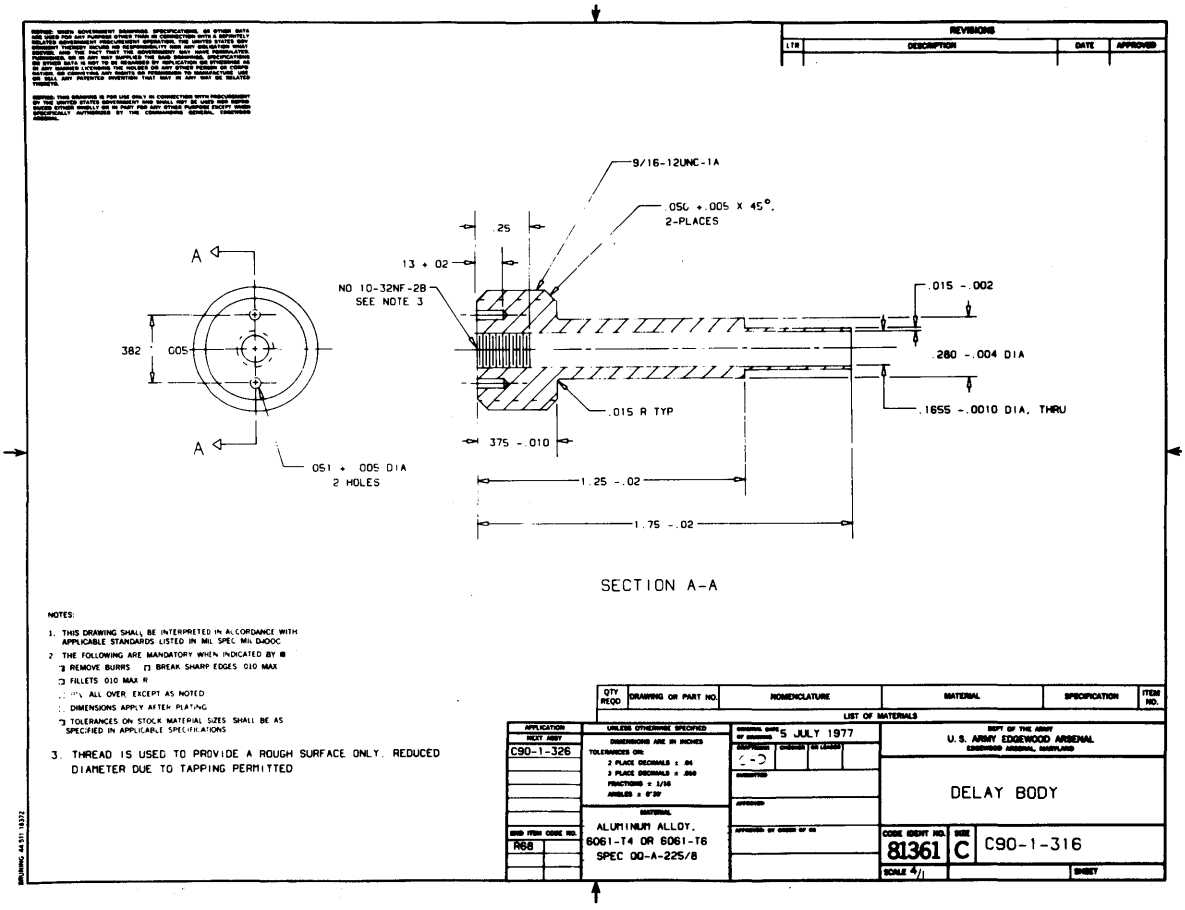


Figure 10—Rocket detail drawing

has continually increased. When the prototype phase is completed, the drawings must be redrawn according to DOD-D-1000 in a level 3 format. This format is required for all engineering production drawings associated with a TDP. The final drawings that become a part of a TDP must be complete and clear so that the manufacturer has no doubt about what he is to produce. Eight prototype drawings originally done on the graphics system were required to be converted to production drawings. The number of drawings was expanded from eight prototype drawings to 22 production drawings to meet the military specification. Drawings had to include subassembly, packing, marking, and individual details on each part (Figures 8, 9, and 10). Careful inspection of the drawings illustrated in the first example with the examples shown above will show the difference between prototype and production drawings. By using the computer-aided-design system, the job was completed in three days. If this work had been accomplished manually, at least three to four weeks would have been required. After the initial drawings were made, an engineering review was conducted which resulted in many additional changes. Using computer-aided-design, all the drawings were made in one day. One week would have been required if done manually.

These examples are only a few of a large variety of engineering applications that can be accomplished through computer-aided design. These examples show simple uses in computer-aided design illustrate that it can be an excellent investment. The average manpower savings generated is about a 4 to 1 ratio. The equipment purchased were both turn-key systems. The drafting and digitizing system cost $250,000 and was amortized in less than 2½ years. The first interactive graphics system costing $150,000 was amortized in 1½ years and the second system, worth $75,000, showed a return on investment in less than one year. With these results in equipment and software amortization, there is little doubt that computer-aided design has become an economic necessity.

TECHNICAL DATA AUTOMATION

The automation of Technical Data is concerned with a means of creating, storing, and retrieving technical data using computer technology. Within the Department of the

Army, there is a requirement to develop and maintain various types of data. This data is in the form of engineering drawings, specifications, standards, packaging sheets, and Quality Assurance provisions. All of this data is currently manually reduced from original drawings or type-written pages to 35mm film and mounted in 80-column aperture cards containing the document's identification. These aperture cards are manually gathered and form the technical documentation required for government procurement.

Computer-aided design and drafting techniques have made it possible to create an engineering drawing on a Cathode-ray tube (CRT). This method has proven to be extremely fast and highly cost effective. It also permits the user to create, store, and retrieve engineering drawings electronically in digital form.

This initial step has formed the basis for the automation of technical data. Data stored in this fashion allows us to go directly to a computer output microfilm unit to obtain the 35mm film which is then mounted on the aperture card. Funds are currently available for such a unit and it is anticipated the system will be purchased and in operation by the end of 1978.

A major problem, however, develops in trying to get existing drawings and various data into the computer system. Manual digitizing is too costly. To eliminate this problem, a method is required to scan the drawing or card, vectorize the data and put it into computer memory. Negotiations have recently been completed between ARRADCOM and Altman Associates, Incorporated for an aperture card scanner. The major problem is to extract the data without destroying the inherent vectorization. This is to be accomplished by the use of a high speed random access combination scanner and laser line follower operating with a dedicating computer (Figure 11). The system is expected to be in operation early in 1979.

A mass memory storage unit will also be procured in the near future to handle all the data requirements involved with the technical data package. It will provide on-line storage

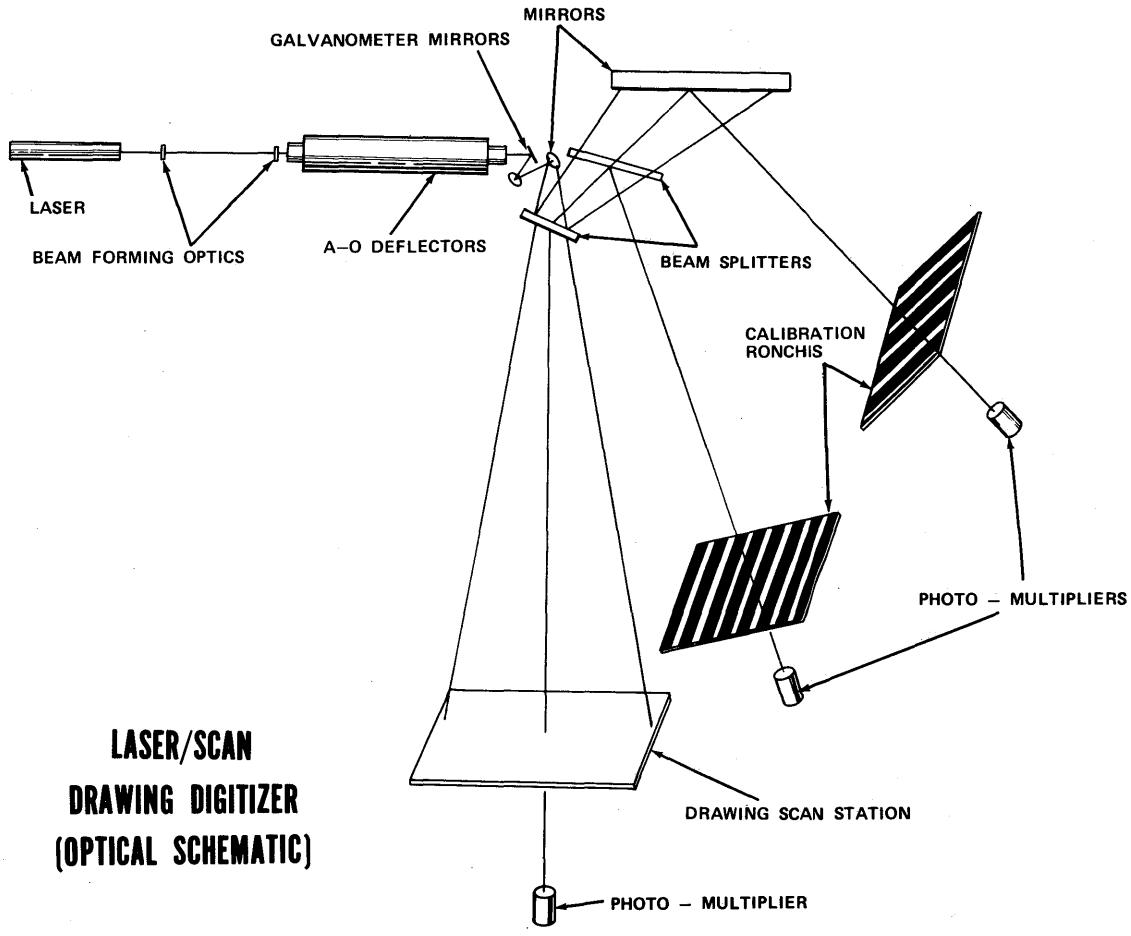


Figure 11—Optical schematic for scanner

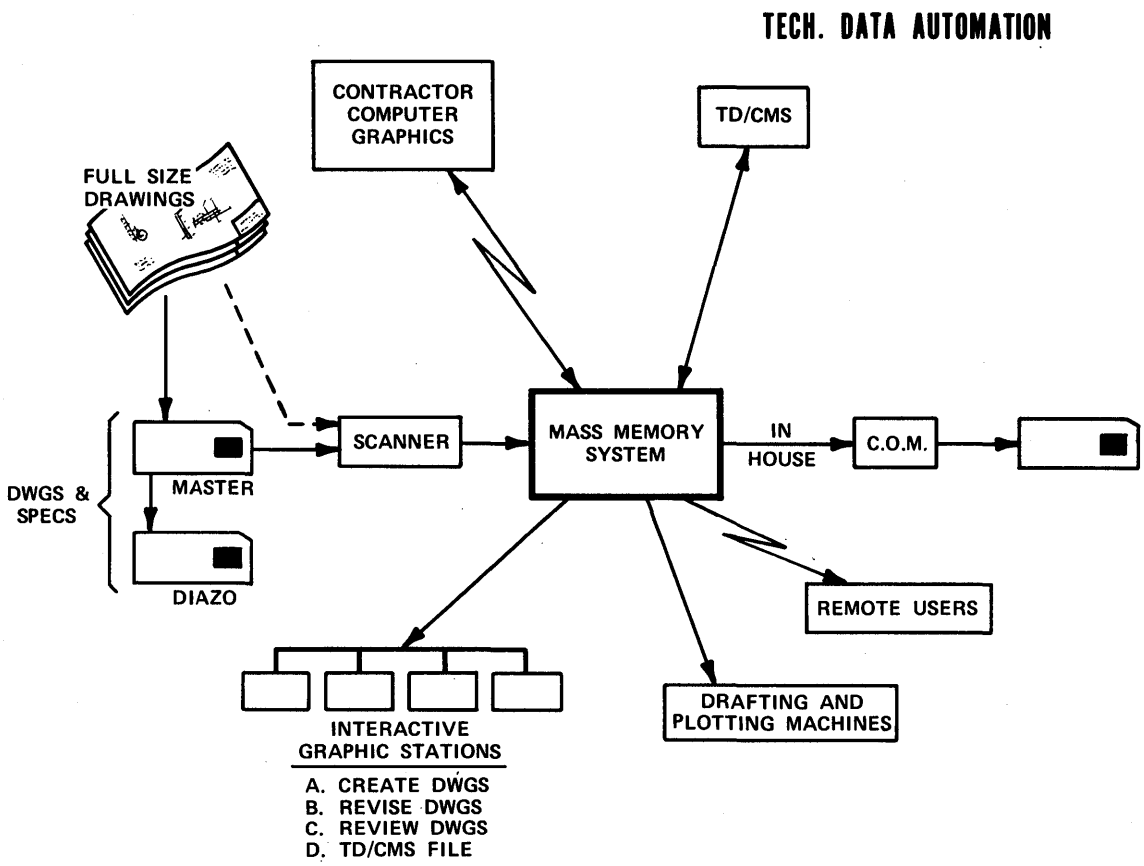


Figure 12—Technical data automation flow diagram

and retrieval capabilities of digital data in archival, non-erasable form and permit random access to any identifiable record.

The combination of this equipment will enable us to change existing drawings or specifications, etc., through Interactive Graphics and go directly to aperture cards without the need of hard copy. The target date for the total system completion is set for 1980. When completed, this technical data automation system will save the government thousands of dollars and greatly reduce the engineering lead time now required in the development of Army materiel (Figure 12).

The Boeing electronic computer aided design system

by BRUCE H. INMAN
Boeing Aerospace Company
Seattle, Washington

PROBLEM

A large percentage of the cost of most DOD-contracted programs is concerned with the concept development, circuit design, packaging, test and fabrication of electronic systems and subsystems. The application of Computer Aided Design (CAD) concepts to facilitate these operations can significantly reduce cost and minimize flow time in the design, fabrication and test of end item hardware.

OBJECTIVE

The fundamental objectives of CAD are to reduce cost and minimize flow times in the design and factory production of electrical hardware. This calls for identification of areas in the equipment design, packaging, test and fabrication processes that are operationally and economically adaptable to computerized techniques and the specification and development of those techniques.

SYSTEM DESCRIPTION

The following is a brief description of the Electronic Computer Aided Design (E/CAD) system in use at The Boeing Company.

E/CAD system description

Some of the factors making the development of electronics systems a lengthy and time-consuming process relate to the sheer mass of detail work that must be performed. Examples are: the drafting and checking of logic diagrams, the tabulation of electrical parametric data (time delays, logic states, voltages, etc.), evaluation to verify electronics designs and functional test sequences, the assignment of logic elements to integrated circuit packages (allocation) and of integrated circuit packages to printed circuit boards (partitioning), placement of circuit packages on the boards, boards in card cages or drawers, and the production of documentation to convey fabrication details to the shop. This accumulation of data during development of a product is an essential procedure, from preliminary design analysis through manufacturing.

To facilitate efficient transfer of data, the E/CAD system was designed to ensure that the programs do not communicate directly with each other but through common groups of data residing in the design data base. This facilitates the addition of new software programs to the system. A separate Data Base Utility program is provided to allow the user to add user-prepared data to the data base and to store and control data.

The Boeing E/CAD System is schematically represented in Figure 1. The rectangular boxes represent programs that are operational on Company computers, while the oval-ended boxes represent manual design functions and are entry points into the system. The programs are divided into two categories: design validation (including analysis, simulation and functional test) and packaging. Analog analysis, digital analysis, Computer Simulator, logic simulator, and digital functional test comprise the design validation programs. They assist the user in converting the data represented by a schematic or logic diagram into a format that can be used by the E/CAD System and provide aids to detect and correct logic errors in the system design. The Logic Simulator program assists the engineer in performing detailed logic simulation (design verification) of digital assemblies, including automatic worst case timing analysis. The Digital Functional Test program aids in the efficient development of high quality functional tests. Also included are post-processing programs which translate the test output data to releasable documentation and format the data for use on automatic test equipment. In addition, numerous analog circuit analysis programs (e.g., ISPICE, CIRCUS, SCEPTRE) and thermal analysis programs (e.g., BETA) are utilized but are not currently integrated into the E/CAD system.

The Packaging Interface program is the interface between design validation and packaging design. User-supplied packaging information (the assignment of logic devices to parts and parts to assemblies) is combined with the design description from the analysis programs. The Packaging Inter-

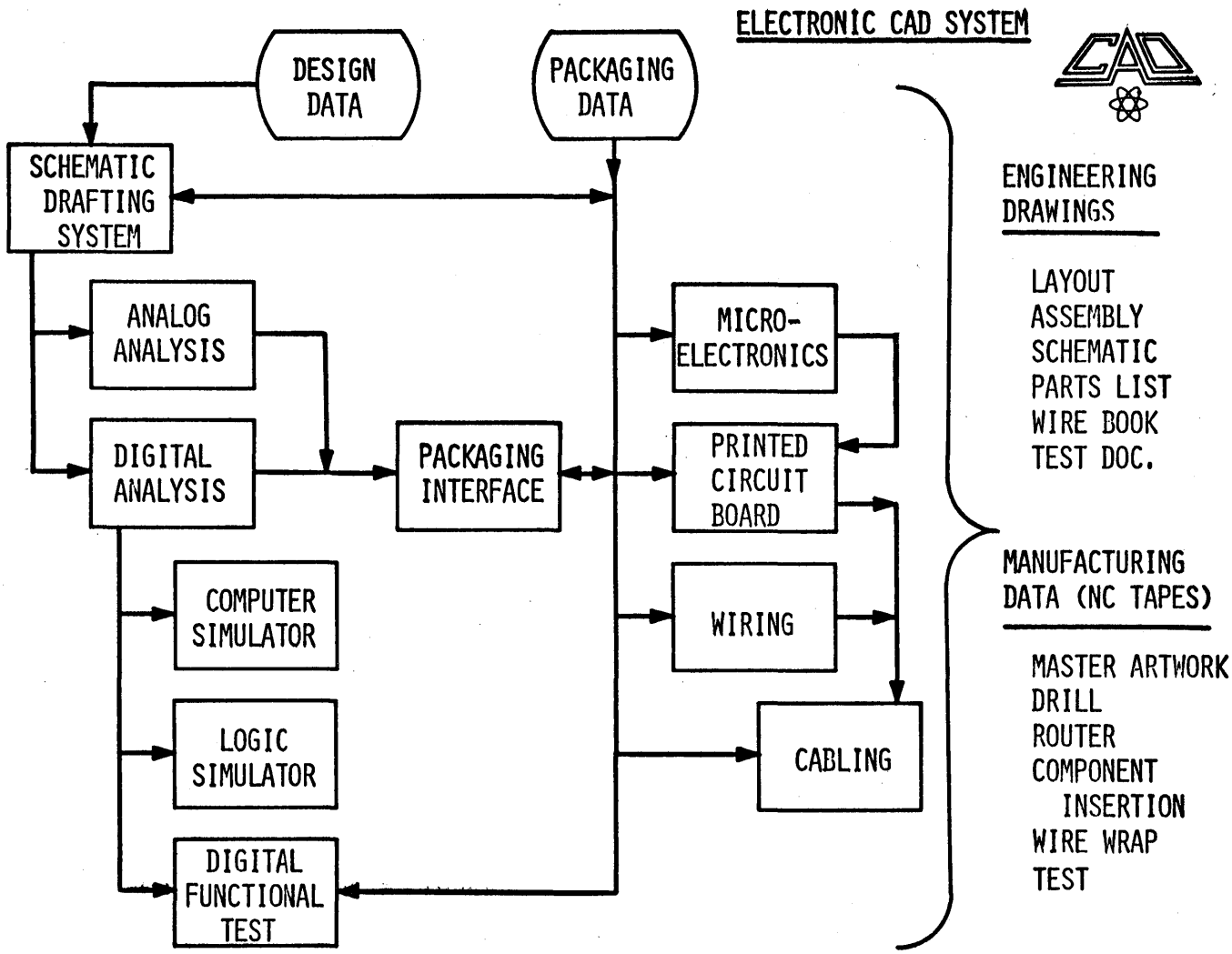


Figure 1—E/CAD system diagram

face program ensures against incompatibilities between logic design and packaging design information, determines all signals that must be brought to the connectors on the assembly, automatically assigns pin numbers for all logic devices, and generates pin-node data for use by the packaging programs and the functional test patterns for producing the final test program package.

The packaging programs are divided into two major categories: printed circuit board design and backplane design. Design of a printed circuit board (PCB) is accomplished with a combination of large-scale computer and small midicomputer with interactive graphics. The PCB editor utilizes packaging design information in addition to the data from the Packaging Interface program. Packaging design information (design rules, pad sizes, spacing, optional part placement, etc) is entered into the system, processed, and edited to correct errors. The resulting information is used for interactive placement of components and automatic routing of conductors. Once automatic routing is completed, options are provided for manual intervention and modification of conductor routes by means of the interactive graphics console (Figure 2). Automatic checking for connection and dimensional spacing violations is a part of the software system. Once board layout is completed, necessary information is generated to drive plotters to produce assembly drawing and photo master artwork and a control tape for numerically controlled drilling is produced. The assembly drawing, photo master artwork, and drill tape, when combined with other manually prepared instructions, are used to produce the finished printed circuit board.

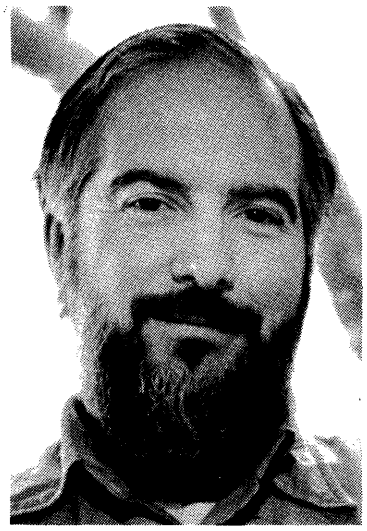
The second major category of packaging programs is backplane design, which involves point-to-point wiring predominantly by wire wrap. This packaging technique is used for two general categories of assembly design: digital logic assemblies and PCB connector backplanes. Wire wrap assembly of digital logic assemblies usually proves to be more cost effective than PCB's in applications such as breadboards and small-quantity production runs. The shop aids

Figure 2—Interactive PCB design

necessary to wire the assembly are produced from the wiring program in the form of a punched tape and a computer listing. This tape and listing are used by the shop for fabrication. The wiring program also places data on the data base for use by a documentation program to produce a complete wire book. The wire book contains necessary information to identify the wiring on an assembly and maintain unit configuration control. The documentation program, in turn, places data on the data base for use by the DITMCO Test-Postprocessor program, which generates a test tape for the DITMCO test equipment to check for open or shorted circuits in the completed wiring assembly.

Utilization within boeing

The E/CAD system has been in development since late 1970. An overall system approach was selected and capabilities added on a modular basis. The E/CAD system is used on all major electronic design and development activities within The Boeing Company. To date, approximately 2,000 PCB's and 700 wired assemblies have been designed using these capabilities. The average PCB is four to six layers and the average wired assembly has in excess of 1,000 wires. Cost savings/avoidance to date are in excess of 10 million dollars.



Area Director
Jim C. Warren, Jr.
Dr. Dobb's Journal of
Computer Calisthenics and
Orthodontia
Menlo Park, California

Home & hobbying computing

OVERVIEW

Computers became consumer products around the beginning of 1975. In less than three and one half years, a considerable array of companies have come into existence to address the new marketplace of "personal computing." This portion of the Conference Program addresses the topic of home and hobby computing from several aspects.

A survey is provided of the brief "history," the current situation, and the foreseeable futures of personal computing. Following this overview, a discussion will address the variety of usual and unusual hardware, software, and systems that have appeared in the personal computing application areas. This will include comments regarding the problems and promise of home computing hardware, software, and applications.

To date, personal computing has been limited to the fifty to one hundred thousand computer enthusiasts who currently own home computers. This is not a mass consumer market, however it is evident that such a market is about to open up, in some ways comparable to the market for stereos, television, and CB radio. The final session in this Technical Area of the Conference Program will address this topic—providing computer power for the average lay person; the mass consumer who will use computers only to the extent that they may be used without expertise, and only to the extent that they can solve problems that the average person may be interested in solving.

THEMES

SESSION 1: Computers became consumer products around the beginning of 1975. Since then, a multitude of companies have been created to address the new

marketplace of "personal computing." There are 50,000-90,000 computers in people's homes, today. The speaker will briefly survey the "history" of this movement, detail the present situation, and hypothesize concerning the foreseeable future. Panelists will address various aspects of the alternative foreseeable futures.

SESSION 2: A variety of usual and unusual hardware, software, and applications are appearing in the world of computers as computers become personal devices. These new facilities provide a wide range of problems and promise. The speaker will provide an overview of the positive and negative aspects of personal computing equipment, programs, and uses. Panelists will present a variety of views regarding the hard and soft facilities that are unique to consumer computing.

SESSION 3: The microcomputer emerged from a laboratory curiosity to a hobbyist toy and from there to a usable business and industrial tool. What of its potential as a mass merchandised electronics device (like radios and televisions) which can be used by persons technically disinterested and incompetent to do other than turn a knob or two?

Representatives from companies which are near the leading edge of this market will present their views and insights in this session.

Personal Computing—A little past and a lot of future

by PORTIA ISAACSON
The Micro Store
Richardson, Texas

INTRODUCTION

We are incredibly lucky! When I look back in time, I see no time more exciting than now! When I look forward in time, I see no time more exciting than now! For now, we are privileged to witness the dawn of the era of personal information processing. Few people have ever lived at such a time of technological change. Perhaps only those who were present when the first book was printed have, before us, had a similar opportunity to observe the birth of a technology-derived tool having the power of the personal computer to improve our quality of life. We are presently witnessing what is probably the most significant of all technology-based revolutions surpassing in importance the printing press, the automobile and the assembly line.

Low-cost abundant computing brings us to the dawn of a new era. A new era in which information processing power will no longer be the exclusive tool of government and large business. Rather we will have computers for people to use in a near limitless variety of ways in our work, our play, and all aspects of our daily lives.

THE COMPUTER HOBBY

Five years ago, and even before, those of us involved with microprocessors were making rather accurate predictions as to the declining cost of computing resources based on our knowledge of the new microprocessor technology. We did not predict, however, the absolutely incredible growth of hobby computing.

Although there had been some activity previously by really enterprising do-it-yourselfers, the computer hobby really took off in January, 1975, when the MITS Altair Computer appeared on the cover of *Popular Electronics* magazine.[1] The computer was available in kit form for about $400. A new hobby and a new industry were born.

In July, 1975, Dick Heiser opened the first retail computer store. Many others were to follow. The computer store started making its place in the computer industry as a new way to market and service low-cost computers and related equipment.

In the summer of '75, MITS was driving its van all over the country demonstrating the Altair computer. In city after city computer hobby clubs were born at the MITS van, which brought the computer hobbyists in each geographic area together for the first time. Now computer hobby clubs number more than 300, with several in other countries. One of these clubs has membership numbering in the thousands and hundreds is very common. Soon these clubs will join together to form an international amateur computer society.

What sort of person is this computer hobbyist? He or she is a hardware or software tinkerer—an experimenter. The computer hobbyist comes complete with clubs, conventions, magazines, T-shirts, and bumper stickers. Hobby computing will surely become the elite of the technological hobbies. Who is this computer hobbyist? He or she is an engineer, a programmer, an electronics buff, an amateur radio enthusiast, a lawyer, a small business person. Most actually have surprisingly serious intended uses for their computers. In fact, a surprising number are depreciating their computers on their income tax returns.

Personal computing is not only hobby computing. The fact that hobby computing has received so much publicity is really misleading. The graph in Figure 1 illustrates this point. Discussions with several computer stores indicate that the personal computing equipment sold over the past year has gone, for the most part, to business, industry and educational institutions. Only about 20 percent of the personal computers sold this past year have been sold to hobbyists. To further illustrate this point, customers of my computer store include: oil companies, universities, hospitals, trucking companies, security companies, public schools, personnel agencies, power and light companies, manufacturing companies, and electronics companies.

Personal computers are replacing time-sharing services in businesses large and small. Personal computers are replacing mini computers in process control applications. Additionally, personal computers are finding their way into new applications where the new low cost of computing makes possible the use of computing where before it was too expensive. One way to view personal computing is that it is simply low-cost computing. And this new low-cost computing will compete with all other traditional forms of computing resources.

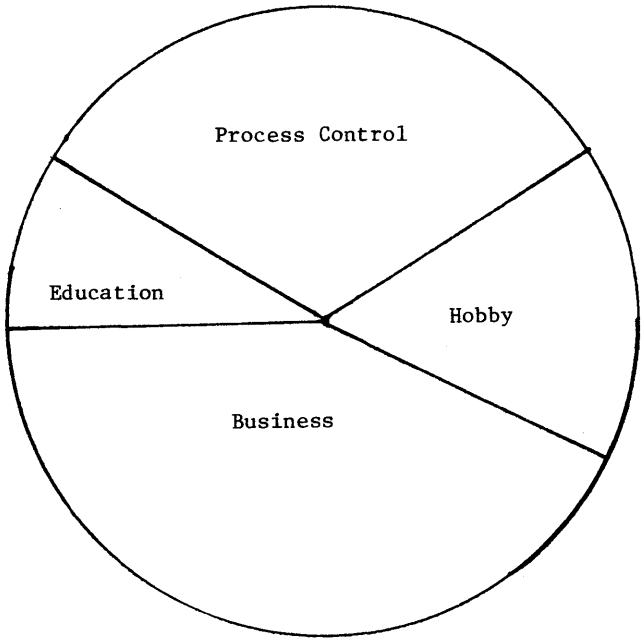


Figure 1—Personal computing sales.

A NEW INDUSTRY

A new industry has emerged to support the personal computing market. There are more than 100 companies manufacturing personal computing products. There are more than 500 computer stores selling and servicing these products. There are 10 major magazines dedicated to personal computing.

The new computer industry differs in several important respects from the computer industry that preceded it. These differences include fierce competition around a standard internal bus and a new way of marketing and servicing computers.

The standard bus

Fundamental to this difference and to the development of the personal computing industry itself has been the S-100 bus standard. The S-100 bus is a set of electrical and logical specifications for the connections between the various printed circuit cards that may share the bus. These printed circuit cards can include: a CPU card, memory cards, keyboard and video-display interface cards, audio cassette interface cards, floppy disk interface cards, music synthesis cards, voice recognition cards, joy stick control cards, and so on. The S-100 bus was originally the MITS Altair bus. Now there are at least twenty companies manufacturing S-100 bus computers. There are more than 150 different products including computers, memories, disks, printers, graphics displays, joy sticks, music synthesis equipment, voice recognition boxes, voice generation boxes, and home electrical system controls presently being manufactured for the S-100 bus. The broad range of available S-100 bus equipment greatly exceeds the offerings of any one manufacturer.

The most important aspect of the S-100 bus standard has been its impact on the evolution of the personal computing industry. Up to this time most of the companies manufacturing personal computing products have been small companies, including many garage-type operations. Few of these companies could afford to manufacture and market an entire computer system and most of them offer only one or two types of boards. The S-100 bus standard has created a marketplace large enough to enable a small company to be successful in the market on a low budget offering only a single board that provides some nice option for a S-100 bus computer.

The small entry cost to the S-100 bus market has made the competition fierce. If a product can be manufactured and marketed for less, it will be. Products are priced very close to the manufacturing cost. This is quite unlike the traditional computer industry which is well-known for prices that have little relation to the manufacturing cost but rather have been priced by looking at what the market will bear.

The computer store would not have been nearly so viable a concept without the standard bus. Virtually all the personal computing manufacturers have been under capitalized, resulting in long waits for products. In many cases stores have been forced to buy essentially the same product from several different manufacturers in order to be assured of an adequate and timely supply. Additionally, since many of the products in a computer store are interchangeable, inventories of particular brands can be reduced. A store can choose a product such as an S-100 bus memory board from among a large number of possible suppliers, using consideration such as delivery time and margin as primary parameters rather than being forced to take whatever the CPU manufacturer cares to offer. This kind of competition has enabled stores to get nearer the kind of margin and service they need.

The retail computer store

The new low-cost computer products require a new way of marketing and servicing these products. It no longer makes sense for a salesperson or field engineer to call on the individual customer when his computing system costs only a few thousand dollars. The retail computer store fills this need. Computer stores offer a place to test drive several computing systems before buying one. Stores try to offer off-the-shelf delivery of computers and associated equipment and software. Computer stores offer advice on configuring a system for a particular application from the wide variety of personal computing products available. Most stores repair the equipment that they sell. Some stores offer hardware and software consulting. Most offer a wide assortment of computer books and magazines. Some stores hold classes in using and programming personal computers. The computer store plays a new and essential role in the new computer industry.

THE HOME COMPUTER REVOLUTION

In late 1977 we saw the first signs of the real home computer revolution. The fully assembled complete computer was offered by two different companies, Radio Shack and Commodore, for under $600. These computers include the CPU, memory, a keyboard, video display, and read-only-memory Basic language interpreter. So now we have the hardware in the right price range for the massive home market, but still missing are the huge libraries of programs that will make these computers really useful. This software will come. Eventually we'll buy programs like we buy stereo records—from a wide selection—where some program can be found to satisfy nearly any taste or requirement. When these program libraries are available, millions of home computers will be sold.

Now what would motivate millions of people to buy home computers? The two primary reasons are entertainment and education.

Do not underestimate the sales appeal of the computer game as an entertainment form. Look carefully at the computer as a new form of personally available, intellectually challenging, interactive entertainment. Of course, the computer offers simple games like pong, target, and guess-the-number. Much more importantly, the computer offers the opportunity for simulated experiences without risk. For example, the well-known Star Trek game provides the player the experience of being a star ship captain—the opportunity to make monumental decisions on which rests the fate of the universe. Other games let us pretend to be a ruler of a country or the president of a company. The simulated experience-type of computer game is a powerful new entertainment form. A $600 price tag seems very small for just this one function of the home computer.

Education is the other major reason computers will be sold to millions. The home computer coupled with well-known computer assisted instruction techniques will offer an interactive personalized form of instruction for the home. CAI programs will be available for instruction in subjects as far ranging as arithmetic drills, French grammar, and driver education. This new home educational medium may eventually have a dramatic impact on our educational systems. When the computer-assisted home education center has evolved to its full potential using such technologies as video disks and two-way electronic communication links to schools, the home could very well become the center for educational acitvity. The school's function would be to prepare instructional materials and certify levels of achievement.

Entertainment and education will surely be reasons enough for many people to buy home computers. Of course, the home computer will have many other uses including: personal financial management, astrological forecasting, keeping inventories of collections such as stamp or beer can collections, centralized control of home physical systems, maintaining address lists, and use as an artistic medium in graphics and music. In fact, the uses of computers are as varied as the individuals who apply them. Applications of the home computer have little limit other than the imagination.

COMPUTER INDUSTRY CHANGES

The new low-cost computing will cause many changes in the traditional computer industry. Some of the changes are starting now. Some will not reach their full impact for years to come. The very structure of the data processing organization will be impacted. Additionally, the individual data processing professional may find his or her role changing significantly.

Personal computers are replacing and are competing with mini computers in low-end applications including process control and small business. In order to understand why personal computers are able to successfully compete with minis in many applications, it must be understood that personal computers are not small—they are just inexpensive. A recent article compared the performance of an IBM S/360 Mod 30, the work-horse business data processing computer of the late 1960's, to the personal computer.[2] According to some parameters the personal computer actually exceeds the S/360 Mod 30 in computing capability.

As personal computers move into the mini computer low-end market, minis will be forced to look upward for more applications. They have already started eyeing the strongholds of the large mainframes.

Personal computers are replacing time-sharing services. Time-sharing was devised as a means of bringing computing resources to the individual. Now that the individual can own an entire computer, there is simply little need to time-share a large computer. The advantages of a 16-user personal computer laboratory over a 16-user time-sharing system were recently documented.[3] They include the fact that the personal computers are half the price of the time-sharing system, the personal computing laboratory can be built incrementally with a very low entry cost, personal computers offer the latest in technology, the personal computing laboratory is more reliable and available more of the time, and the individual can typically have more disk storage space. Disadvantages of the personal computing laboratory include the lack of large software libraries and the fact that very large programs that might use the entire time-sharing computer during non-time-sharing hours cannot be run. In many cases the personal computer offers a better way than time-sharing of bringing computing resources to the individual.

The data processing center will soon find itself loosing control of the corporate data processing function. This has already happened to a certain extent as mini computers first meant for process control applications were found useful as general purpose data processing machines. Departments within corporations found that by buying their own small computer they could spend significantly less on data processing than if they bought computing services from the data processing center. This trend will be greatly accelerated by the still much lower cost of the personal computer. The eventual result of this type of pressure on the centralized

data processing facility will be pressure on the large mainframe manufacturer to lower prices to compete with the low-cost computers. Ultimately personal computers will force lowering of prices by the large mainframe manufacturers.

A major change for the computer professional will be the fact that the public will be computer literate. The public will have a much better understanding of computers and software. No longer will they be awestricken and unquestioning about a computerized process. They will understand the role of the programmer and demand that programs interfacing with the public offer reasonable human interfaces and consideration.

Today society's view of the computer programmer is with considerable awe and respect. Other departments in a corporation view the data processing person as the practitioner of the black art of computer magic. Most of us in computing have at one time or another used our technological jargon to our advantage in cutting short a discussion. All this will change when computing and programming become household activities. The computing profession will not be viewed with quite the splendor it once was. This author occasionally reflects on the fact that a Ph.D. in Computer Science will have the same social prestige as a Ph.D. in refrigerator science when the computer becomes just another household appliance.

It is well-known that one of the biggest problems in the computer industry today is the high cost of software. Low-cost computing may eventually help solve that problem in an indirect way. Low-cost computing will lead to widely available computer education at all levels. Now that the cost of a computer is not much different than the cost of a typewriter, the major block to widely available computer education is gone. With computer education abundant there will be a flood of qualified entry level programmers. The effect may ultimately be more competition for data processing positions and ultimately lower salaries for programmers. The result: a decline in software costs. A possible weakness in this scenario is that at the same time the number of computer trained people is increasing, the amount of software needed by new low-cost computing applications is also increasing. Perhaps the need for programmers will be adequate to absorb the higher production.

CHANGES IN OUR WAY OF LIFE

Predicting the future is always a risky venture at best. However, we can take a few peeks at what the effects of abundant low-cost computing may be. We have already discussed the fact that to our homes the computer will bring powerful and exciting new forms of entertainment and education.

For those skeptical about the magnitude of the potential impacts of the home computer a simple scenario usually removes any doubts. What if a million people, or a thousand, or even just a hundred all owned the same type of computer; all used the same stock market analysis program, and—based on predictions of the program—all made the same moves in the market? Chaos!

Probably the largest contributing factor to change is that now computer-inventiveness is in the public domain. No longer can only large business and well-endowed universities invent devices having computers in them, computers themselves or computer-related equipment. Now because of the new low cost of computing, the average person can use computers as components in inventions giving them a new dimension—intelligence. History tells us that many of our most personally useful inventions were not invented by large corporations or universities, but rather by an individual with a problem to solve. Now that same kind of motivation and inventiveness will be applied to computer products. In fact, from the present computer hobby movement will come gizmos too numerous to mention. We are now on the threshold of a gizmo age in which we will be surrounded by intelligent devices supplying services that would be impossible to imagine at this time.

All is not good, however. For example, it is now possible to build a computer-based device that will place telephone calls, play a recorded message, record the verbal response, and even accept the touch-tone input of a credit card number. This device can be very persistent and devious in placing calls. For example, it can either place calls against a provided list of numbers or it can generate the numbers within a given prefix which, of course, corresponds to a geographic area. Notice your unlisted telephone number is of no help. The call-placing device can remember that you didn't answer and place the call periodically until you do. It could even remember that you hung up! The devastating truth is that a device can be built to place junk telephone calls for about $2\frac{1}{2}¢$ per call compared to $7\frac{1}{2}¢$ per letter in postage alone for junk mail. The only answer is probably legislation. In the meantime a low-cost device can be built that will answer your telephone and require entry of a touch-tone password before the household is notified of the call.

Over the long range the abundant computer will add a new and rewarding dimension to our lives. We have a new fundamental tool, a new medium for expression and experience. We are indeed incredibly lucky to be present at the dawn of the era of personal information processing.

REFERENCES

1. Warren, Jim C., Mark E. Deppe, and James P. Fry, "Personal Computing—An Overview for Computer Professionals," *AFIPS Conference Proceedings*, Vol. 46, 1977, pp. 493-498.
2. Isaacson, Portia, *Datamation*, Personal Computing Department, November, 1977.
3. Isaacson, Portia, *Proceedings of the 1977 Computer Users Conference*, Computer Science Department, East Texas State University, Commerce, Texas, March 25, 1977, pp. 59-68.

Personal computers—Hardware, software and documentation

by JEF RASKIN
Brisbane, California

INTRODUCTION

What is a personal computer? Possible candidates for personal computers range from programmable calculators and games on one end to an antique IBM 7094 in a hobbyist's basement on the other. There is no doubt that the pocket sized programmable HP-65 and its successors *are* computers in the classical sense. They can be programmed to execute any algorithms that fit within their memory limitations. This is all that can be said of *any* computer. For this discussion, a useful dividing line between calculators and computers is that calculators do not provide for the input and output of natural language text.

It is more difficult to draw a line between *personal computers* and other devices that are correctly called computers. In fact, there is no special term for those computers not intended to be personal computers. The term "impersonal computers" is tempting. For this discussion (and at this point in history) personal computers will be defined as those computers that are *marketed* as personal computers. This correctly reflects the fact that there is nothing special about personal computers except that they are inexpensive enough to be reasonably sold to large numbers of individuals.

Excluded from this discussion are truly home-brewed computers, designed and built by a rugged individual for his or her own edification and amusement. Also excluded from this discussion are computers, however small, marketed as small business systems. They may, in fact, be identical to the systems that are being discussed, but very different considerations apply to their software and documentation—they are also directed toward a different audience. Business users will apply different criteria than home users to the selection of a system. During business hours, at least, they are not personal computer systems. It is less than reckless to predict that they will have their own publications and conferences in the near future.

The systems considered here are those that are being programmed and used by thousands of people, each of whom can point to the computer and say, "It's mine."

HARDWARE

A personal computer is based on one of the commercially available microprocessors. Most begin with an 8080 or a 6800, or their successors (such as the 8085, 280 or the 6500 series). The hardware achieved commercial quality very quickly, except for a few design errors, some of which have become almost legendary. By and large, personal computers are reliable. They use the same quality components, boards connectors, enclosures, and manufacturing processes as commercial computer products—with only a few brands cutting corners.

PERPHERALS

Anybody familiar with minicomputers might expect that paper tape readers and punches would be major peripherals in micros. It is not so. Cassette tape machines and ordinary commercial audio recorders dominate the field. Simple but clever interfaces make them moderately reliable off-line storage devices. There used to be a certain fascination in buying your computer peripherals on special at a discount store. Since you will soon be able to buy the whole computer system there, the novelty will quickly wear off.

One of ten best personal computer peripherals is the memory-mapped video display. The speed and capabilities of these displays—which drive an ordinary TV monitor—rival and often best even the more expensive "intelligent" terminals. Their main limitation is their small (typically 16 line by 64 character) display. Many TV drivers display fewer characters than that. Nonetheless, most *mini*-manufacturers have far poorer console displays. Most of the personal computer displays provide some form of graphics as a standard feature; a few brands even offer color graphics.

Floppy disk drives and controllers do not differ significantly from the rest of the industry. Hard disks for personal computers are not yet available, and may never be; it all depends on the price of various new forms of memory. Again, the personal computers are in the same position as the rest of the industry. Keyboards are mostly encoded, with a few more advanced systems going to polled operation. Keyboards are rapidly being integrated into products.

Paper tape is used. The only popular punch is the ASR33. Some clever tape readers have appeared, including at least two that can go as fast as you can pull the tape through. Since they work asynchronously, all one needs is a simple electric motor and a spool to have a cheap and remarkably fast tape reader.

Punched cards have no hold in the personal computer industry at all.

The number and variety of peripherals and interfaces for personal computers are incredible. Just one system the popular "S-100" bus, has over 300 different devices that are mechanically and electrically (more or less) compatible. Never has one bus been available on so many different computer brands. Never have so many *hundreds* of companies made accessories for the same bus.

MEMORY

It took the personal computers to show the industry how cheaply memory could be sold. The personal computer designers have not been shy at using the latest memory chips as they have become available. In hardware, the personal computer user has the latest, the best, and the cheapest.

SOFTWARE

Personal computer software, for the most part, does not attain the state-of-the-commercial-art level that the hardware does. There is a unique problem in the hobbyist industry: almost everybody has a choice of only a few processors. In the mini and maxi computer world, manufacturers could afford to develop software, since it could only be used on their brand of computer. This problem has only begun to creep into the mini and maxi computer world. At present, it is not clear that a substantial investment in software development can return a profit. Selling packages to individuals at $5.00 to $50.00 a shot is working for some small, low-overhead organizations and individual free-lance programmers. Writing the obligatory BASIC interpreter for various companies is keeping more than one firm alive.

Very few owners of personal computers are aware of the benefits of the more advanced computer languages, and therefore, are content with an endless stream of ad hoc additions to BASIC. Since the structure of BASIC makes writing long and complex programs lugubrious, programs tend to be short. Since the overhead for remarks is high, they tend to be poorly documented at best.

One of the most widely used assemblers for the 8080—you may not believe this—is missing a few instructions! Assemblers that offer macros and relocatable code are only beginning to be available. Noise about APL is often heard, but the product lags far behind. FORTRAN is beginning to be used, and there is some interest, for some reason, in COBOL. FORTH is available but overpriced for most of the market.

An amazing software development has been the versions of Tiny BASIC. They use interpreters whose object code can easily be displayed on a single quarto-sized page. The cheap personal computer has forced such development on one hand, and made it usable on the other. The disk-operating systems are rapidly gaining in sophistication. Personal computer software is a funny mixture of cleverness and conservative copying of outmoded programming techniques.

DOCUMENTATION

Personal computers are being marketed mainly at first-time users. That is not the current user population—that is the *anticipated* user population. With few, if any, exceptions there aren't any computers systems that you can hand a beginner, then come back a week later and find the person happy and the computer up and running. The fault is partly with the software and mostly with the documentation. This will have to change radically for the better in the next few years, or personal computers will continue to be the exclusive toys of the electronics engineers and programmers who now buy them.

Area Director:
John McLeod
SIMULATION
La Jolla, California

SIMULATION

An increasingly important application of computers is their use for simulation. Simulation involves performing experiments on models instead of the real-world system—the simuland—for obvious reasons. Simulation is cost-effective whereas experimenting with real systems is usually more expensive, may be dangerous, and is often impossible because the simuland is not available, or is intractible to experimental manipulation.

In recognition of the current and exponentially increasing importance of the use of computers for modeling and simulation the NCC Program Committee has dedicated a Program Area to the subject.

There are three sessions in this Simulation Area. One is a panel discussion during which panelists will address the question "Can Large-Scale Models Alleviate World Resource Problems?" The written Position Statements of each panelist has been distributed to all other panelists with the expectation that each will thus be prepared better to discuss points raised by the others. However, besides setting the stage for a more enlightening discussion, this was done because panelists were selected to some extent because of their differing points of view and it was thought "forewarned would be forearmed" with pertinent material.

An effort was made to keep the number of panelists relatively small so that each could have a reasonable time in which to present his views, then defend them if they are challenged either by other panelists or by members of the audience. On the other hand the panel organizer and moderator has invited knowledgeable (and sometimes opinionated) simulationists of his acquaintance to join in the discussion from the audience.

In addition to the panel session there are two simulation sessions for the presentation of technical papers. The subject of one of these is:

MICROPROGRAMMING AND SIMULATION

This session was arranged by Dr. Lance A. Leventhal, Society for Computer Simulation, in cooperation with Dr. Gary Nutt, Department of Computer Science, University of Colorado, Boulder, Colorado, the Session Chairman.

The concept for this session was developed because new technical developments have extended the range and power of simulation. One long-standing problem however, has been the development of software and systems for computers which are unavailable for use, unsuited for development, or not yet even built. Simulation has long been used to alleviate this problem while allowing experimentation with new architectures. Recent hardware developments have made it possible to create special computer facilities for these purposes. Such facilities can serve as universal hosts, capable of simulating (or emulating at the microprogram level) a wide variety of computers. One paper in the session on "Microprogamming and Simulation" presents an overview of the uses and advantages of emulation. A second paper describes an emulation facility while a third paper discusses an example application.

These papers are:

"Emulation: Tool for Software Development," by N. F. Schneidewind, Department of Computer Science, Naval Postgraduate School, Monterey, CA. 93940;

"PRIM System: a Framework for Emulation-based Debugging Tools," by J. Goldberg, A. Cooperband, and L. Gallenson, University of Southern California Information Sciences Institute, Marina Del Rey, CA. 90291, and

"A Microprogrammed AN/UYK-20 (V) Emulation," by D. A. Deel and W. A. Burkhard, Computer Science Division, Department of Applied Physics and Information Science, University of California, San Diego, La Jolla, CA. 92093.

The subject of another session is:

PERIPHERAL AND MULTI-PROCESSORS IN SIMULATION

Also arranged by Dr. Lance Leventhal, Society for Computer Simulation, this one with the cooperation of Dr. Walter J. Karplus, Computer Science Department, University of California, Los Angeles, Session Chairman.

The fact that hardware can now be inexpensively produced in configuration specifically suited to simulation applications inspired this session. This is important because most simulation problems involve a large number of concurrent activities and hardware elements can be connected in parallel to handle such problems. Several different approaches are possible; the parallel system may be an add-on to a general-purpose computer, a general parallel multiprocessor may be built, or a parallel system may be specifically configured for a particular class of problems. One paper in the session on "Peripheral and Multi-processors in Simulation" describes a system that can be used with many large computers. Two other papers describe the use of multiple processors working in parallel to solve plasma and fluid flow simulation problems.

The papers in this session are:

"A Multiprocessor Digital Computer for Dynamic Systems Simulation" by R. M. Howe, Dept. of Aerospace Engineering, University of Michigan, Ann Arbor, Mich. and E. Gilbert, Applied Dynamics International, Ann Arbor, Mich. 48104;

"Plasma Simulation on the UCLA CHI Computer System" by R. W. Huff and C. C. Wu, Dept. of Physics, University of California, Los Angeles, Los Angeles, CA. 90024; and

"A Multiple Microcomputer Approach to Fluid Flow Problems" by J. Steinhoff, Research Department, Grumman Aerospace Corporation, Bethpage, N.Y. 11714.

Emulation—Tool for software development

by N. F. SCHNEIDEWIND
Naval Postgraduate School
Monterey, California

INTRODUCTION

A major computer operations problem is the conversion of programs from one language to another when a replacement computer is acquired. Emulation was developed as one solution to the conversion problem. Emulation allows the machine instructions of the emulated (target) machine to be executed on the emulating (host) machine. Thus permanent program conversion is avoided. Frequently, emulated programs execute several times faster than their non-emulated counterparts. This occurs because the host is faster than the target or its microinstructions are more efficient than the target's machine instructions, or for both reasons. Other methods of program conversion which compete with emulation are: recompilation, reassembly, translation and simulation (interpretation).

Although "translate" is frequently used as a general term to denote all types of language conversion, including assemblers and compilers, here it will be used more specifically to mean a conversion from language A to language B where A and B are not machine languages. It should be noted that recompilation, reassembly and translation effect a permanent conversion, i.e., the result of the process is a program in another language, whereas emulation and simulation employ another language, microinstructions and interpretive routines, respectively, only during the period of program execution. Another way of viewing this difference is that execution of a converted program using recompilation, reassembly or translation is a two step process-conversion followed by execution—whereas emulation and simulation are one step processes; as far as the user can discern, conversion and execution take place simultaneously in a single computer process.

Another application of program conversion techniques is in a software development organization, where a multiplicity of hardware and software systems are produced. This occurs, for example, in organizations which are developers of tactical software systems for ships and aircraft. Typically, many computers comprise a single system. The variety of computers is even greater across the many systems with which the software development organization could become involved. It would be quite expensive for the software organization to acquire all the required computers and programming languages, particularly in view of the transient nature of the development process. In addition, the computers and languages which are needed for testing purposes are frequently unavailable at test time. If the organization were to wait until all hardware and software is designed and implemented, system integration would be delayed to the point of causing delivery slippage. Thus a procedure is necessary for executing and testing programs prior to the availability of the hardware which will be used in the delivered system. In addition to the hardware being unavailable, it is possible that compilers to be used with the hardware are also unavailable at test time. This situation would require a method for producing programs which are not dependent on the use of machine language generated by the operational compiler. Here and subsequently, "operational" will refer to the software and hardware which will be used in the field after the system is delivered to the customer.

This paper will describe and evaluate different alternatives for providing a software development organization with the capability of writing and testing programs prior to the availability of the operational hardware and compilers. In this context we will describe the concept of a software production and test facility which could be used for the development and implementation of a variety of software and hardware systems in the absence of full operational capability. We will also address some of the problems which arise in the execution of this concept, such as the difficulty of faithfully representing I/O operations.

PROGRAM CONVERSION ALTERNATIVES FOR A SOFTWARE DEVELOPMENT ORGANIZATION

We now explore various alternatives for achieving a multihardware, multilingual capability for a software development organization. First, some terminology and notation are in order for the purpose of identifying the various types of languages and machines which may be involved in the operation of the software facility. The terms "target" and "host" are frequently used to refer to the "executed on" and "executed by" machines, respectively. In order to describe the software development environment referred to above, it is necessary to expand the terminology to include two types of languages and two types of machines as shown below. It should be noted that the case described below differs from the situation where an organization converts

from system A to system B. In this case A and B represent the target language/machine and host language/machine, respectively. In the software development organization case, the target language/machine is the operational system and the host language/machine is the system used in the software development facility for testing and integration.

- Source language: Language used for coding the program initially.
- Host language: Language used for executing the source language program at the software development facility.
- Host machine: Machine on which the source program is executed in the software development facility.
- Target machine: Operational computer.

These relationships are shown in Figures 1 and 2. Figure 1 shows the general concept of the software development facility wherein two major choices exist for producing software for test and integration purposes. One method in Figure 1 involves producing host machine language which is executed on the host machine. This method tests problem logic in terms of host machine language. This method could be implemented by selecting one of the following paths in Figure 2:

- 1, 2 Interpretation
- 1, 3, 9, 10 Translation
- 1, 9, 10 Compilation
- 5, 9, 10 Assembly

The second method shown in Figure 1 involves producing target machine language which is executed in the host. This method tests problem logic in terms of target machine language. This method could be implemented by selecting one of the following paths in Figure 2:

- 1, 4, 7 or 1, 6, 7 Compilation followed by Simulation
- 4, 7 Simulation with Assembly Language Input
- 6, 7 Simulation with Machine Language Input
- 4, 6, 8 Assembly followed by Emulation
- 1, 6, 8 Compilation followed by Emulation
- 6, 8 Emulation with Machine Language Input

For tactical system development, the use of either method would be limited to testing the logic of individual programs and the logic of integrated programs. Tests of system timing, storage utilization and hardware dependent tests—performance testing—would be deferred until the target system is available. However, a substantial portion of the testing of most systems involves the checking of sequence of operations and the correct production of outputs for given inputs. The correct functioning of many of these operations is not speed or hardware dependent. This approach is also compatible with the top-down method of system design which emphasizes the early development of the system framework—calling sequences and communication paths. Thus logic testing performed in advance of performance testing will facilitate the entire test process.

In Figure 2, many alternatives are listed for the sake of completeness; some of the alternatives would be impractical for a software development organization to implement. Although many alternatives are shown in Figure 2, only one or two would actually be used in a single organization. We now examine each of the alternatives shown in Figure 2; before doing this, the languages indicated in Figure 2 will be identified.

Item 1. HLL1: A host compiler exists for this high level language.
Item 3. HLL2: A host compiler and translator exist for this high level language. A host compiler does not exist for HLL1.
Item 4. AL1: Target assembly language.
Item 5. AL2: Host assembly language.
Item 6. ML1: Target machine language.
Item 9. ML2: Host machine language.

Interpretation

- One step process.
- Slow because of software approach used in interpretation.
- Does not execute target machine language.

Translation

- Two step process.
- One hundred percent translation is not feasible.
- Does not execute target machine language which is necessary for testing equipment oriented operations.

Compilation and Assembly

- Two step process.
- Does not execute target machine language. (see Translation above)

Simulation

- One step process.
- Slow because of software approach used in simulation. About one-half the speed of the target system.[1]
- Executes target functions but by using host machine instructions to simulate target machine instructions: modified target execution.

Compilation followed by Emulation

- Two step process.
- Relatively easy to program.
- Execute microinstructions in host which carries out target machine instructions.
- Fast execution relative to simulation (at least as fast as target systems).[1]

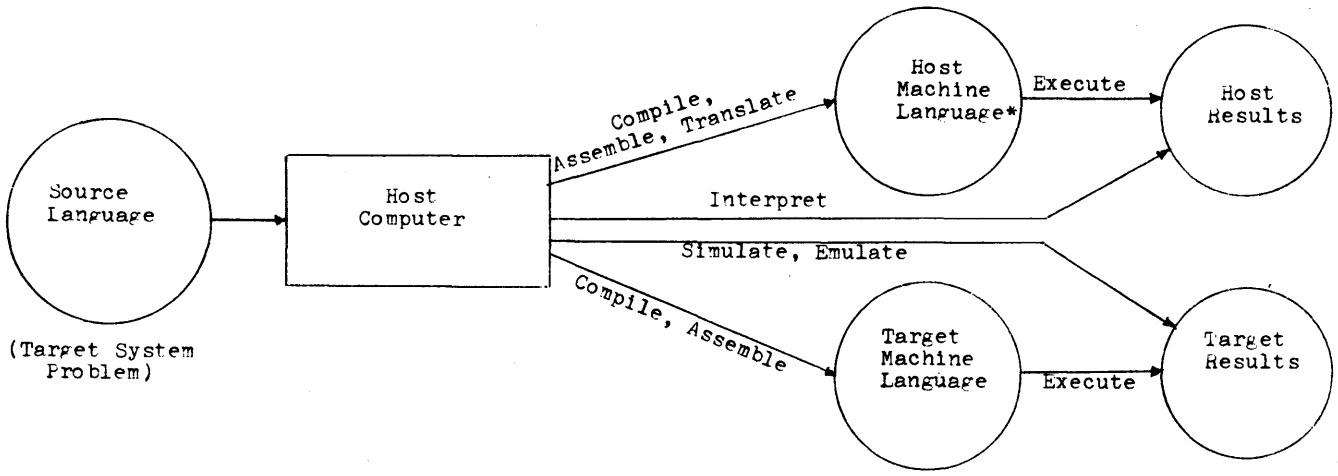


Figure 1—Software development concept

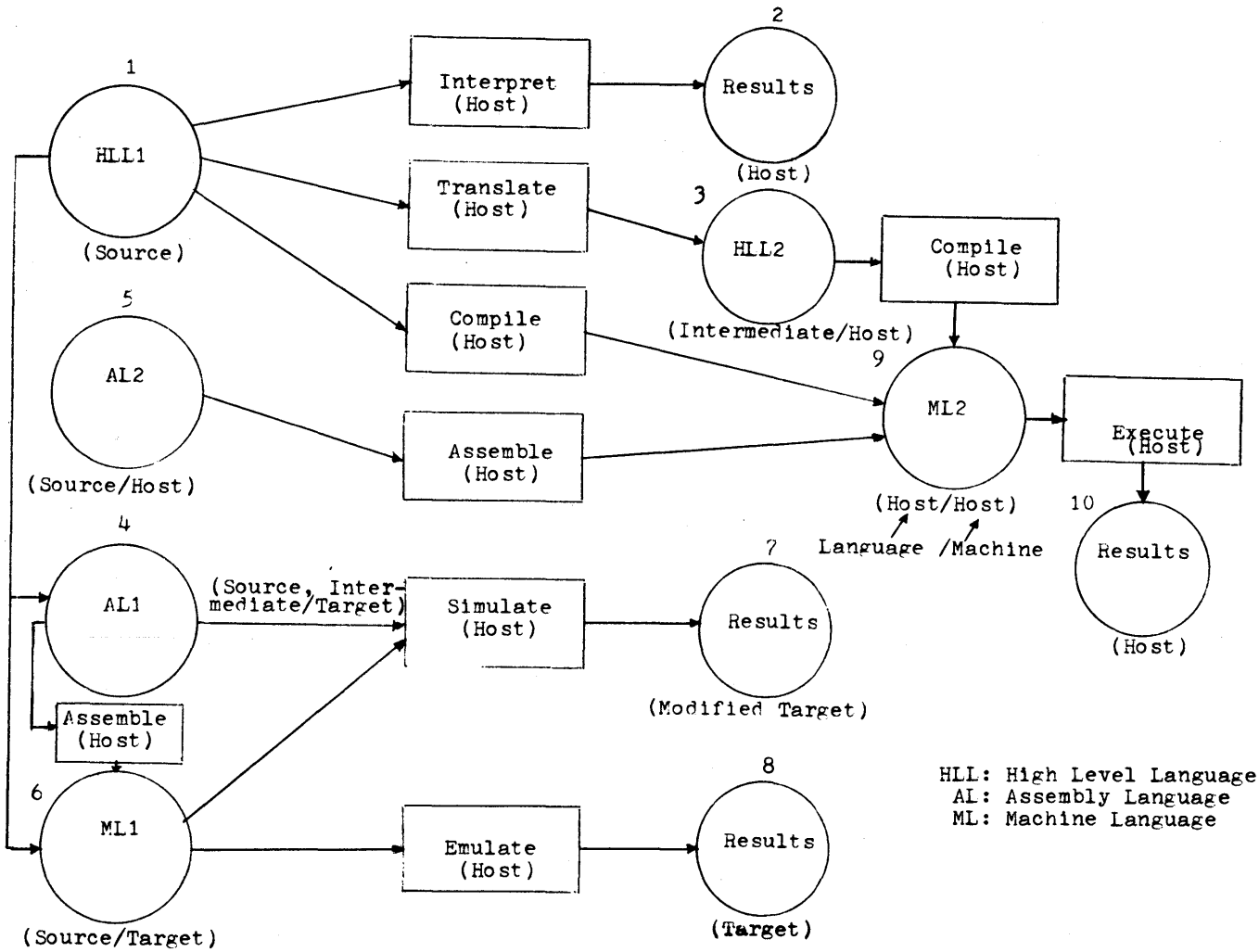


Figure 2—Alternatives for testing target machine programs

Emulation with Machine Language Input

- One step process.
- Difficult to program.
- Execute microinstructions in host which carries out target machine instructions.
- Fast execution relative to simulation (at least as fast as target system).

EVALUATION OF EMULATION FOR A SOFTWARE DEVELOPMENT FACILITY

Some of the considerations involved in deciding whether to employ microprogramming (emulation) for software development are the following:

Word Length

Is the host machine microinstruction length an integral multiple or divisor of the target machine instruction?

Program Size

Is the program so large that target assembly or machine language programming would be prohibited—necessitating the use of a HLL and its compiler for producing machine language which can be emulated?

Control Store Speed

How fast is the control store of the host relative to its main memory speed? Unless the control store is several times the speed of the host's native mode, the speed penalty will be excessive and the target machine's timing characteristics become distorted.

Type of Control Store

Will the emulator be stored in Read Only Memory (ROM) or Writable Control Store (WCS)? The latter is clearly preferable for a software development organization. One computer could suffice for emulating a variety of target machines by changing the contents of WCS. An on-line library of target machine emulation programs, accessible to the WCS, can be envisioned.[2] Another advantage of WCS is that it provides a user microprogramming capability.

Internal Data Code

Does the internal data code in the host differ from that used in the target (byte versus word orientation)? Does one machine use variable and the other fixed word length?

Arithmetic Operations

Do the arithmetic operations differ (decimal versus binary instructions)?

Instruction Length and Format

What are the differences in instruction length and format (size of operation code, number and length of addresses)?

Nature of I/O Operations

Are the I/O operations of target and host vastly different, such as the use of programmed I/O in one and DMA in the other? Even a common device like a card reader may cause problems if the number of columns read, data code or error processing differ between machines.[3] Another example of the complexity of the microprogram which can ensue from attempting to emulate another common device are the control signals of a typewriter terminal—carriage return, margin release, backspace, paper advance, etc.[4]

Microprogram Preparation Tools

How easy will it be to prepare emulation programs? Since microprogramming by conventional means requires a high degree of programming skill, lacks good self-documentation procedures and is difficult to debug, research has been under way to develop high level languages for producing microcode.[5]

Despite some of the problems of using emulation cited above, it can provide a powerful capability for the representation of a variety of systems in a software development organization because of its fundamental property which allows the host architecture to be changed in accordance with different target machine requirements.[6] Rather than provide n sets of hardware in a software development organization, n ROMS can be provided or, better, n emulators can be recorded in a library for loading into WCS. Another alternative is to use a combination of ROM and WCS, where the former is used as a backing store to hold emulator programs and the latter is the working storage where the current emulator program is stored; programs are swapped between ROMS and WCS. This capability would allow testing of a multiple computer target system.

Examples can be cited of the successful employment of emulation in software development.[7] However, in order for emulation to be considered a general solution to the software development problem, its limitations must be assessed and circumvented. Suggestions for dealing with the limitations are discussed in the next section.

LIMITATIONS OF EMULATION

The following discussion addresses certain limitations and issues which are crucial to the success of an emulator oriented software development facility.

Input/output operations

As stated previously the objective of advanced testing in the software development facility is limited to checking for correct operation and sequence of various functions; testing of timing relationships would be deferred until the target hardware, including I/O, is available. In other words, advanced testing checks for correctness of logic; later testing checks for correctness of performance. Using this approach, the following emulation control of I/O could be employed during advanced testing: Upon decoding an I/O command in the emulator program, control would be transferred to the host operating system where a timer would be set to simulate the duration of the I/O operation. Control would then be returned to the emulator. At the expiration of the timer interval, a timer interrupt would occur. For a read, the operating system would load the input buffer from the test data area in memory after the interrupt and prior to returning control to the emulator. For a write, the emulator would transfer data from the test data area to the output buffer; when the write command is decoded, control is passed to the operating system as described previously. In neither case is data transferred to or from a peripheral unit. Only the movement of data in memory, to or from the I/O buffers, is simulated. The time interval which is set in the timer would be drawn from a probability distribution which is representative of the characteristics of the target I/O device.

Speed relationship between host and target machines

It is important to be able to predict the speed of the target machine based on the speed of the host emulator. The speed analysis would be separated into two parts: CPU and I/O. In the case of the CPU operations a time ratio would be established based on sample measurements of time taken in the host to emulate specified target commands and the known time required for executing the target commands. This ratio would be applied to the number of commands to be executed by the target machine in order to estimate its CPU time. Several ratios might be needed, one for each type of command. The I/O timing analysis would be treated differently. Measurement of emulator time would not apply, since the emulator is not involved in I/O operations. Input/output time would be estimated from target I/O equipment specifications and problem specifications: numbers of file accesses, records read/written; blocking factor, etc.

Validation of emulator program

There is no way to guarantee total correctness of emulator operation during advanced testing because all target equipment is not available and performance characteristics involving timing dependencies and storage utilization cannot be totally tested during this period. The best that one can do, as with any test, is to compare the observed operation with system and program specifications. In the case of advanced testing, the comparison is limited to checking of: expected outputs for given inputs and expected sequence of function execution. The time of these events and efficiency of resource utilization cannot be evaluated.

Figure of merit for fidelity of emulation

It is desirable to have a simple-to-compute figure of merit which provides an overall measure of the closeness between emulator and target machine operations. A single comprehensive measure may not be possible. If closeness is to be the criterion, speed ratio is not a good measure, because we desire the host to operate several times faster than the target machine. This would be advantageous for advanced testing because the time and cost of testing would be reduced. A more appropriate measure would account for host deviations from target operations. Examples would be the frequency of simulating I/O. A ratio could be computed of software simulated commands to total number of target commands processed in the host. This would provide a measure of lack of fidelity between emulator and target machine.

Simulation versus emulation

A natural competitor to emulation is simulation. Simulation has an advantage of being less limited in the types of target operations which can be performed. This is because, in general, software provides a flexibility for performing complex operations which is unattainable with microinstructions. An example is the operating system task management and resource allocation operations which are necessary for executing the microprogrammed application functions. However, if all target functions were performed by simulation, there would be a lack of validity in target program testing because the simulator executes host machine instructions. This amounts to executing in pseudo target machine language. In contrast, although emulated CPU instructions are not pure target machine language, the use of microinstructions for representing target machine control and execution sequences comes closer to imitating operational conditions. In addition, emulators are reported to have a performance of five to 10 times more than simulators.[8] The best approach is to blend emulation and simulation so that the characteristics of each are used to advantage.

CONCLUSIONS

Alternatives for providing an advanced testing capability in a software development organization have been discussed. The need arises because some organizations are responsible for developing a series of systems each of which may be comprised of several software and hardware systems. It would be too expensive for the organization to acquire all the hardware and software systems necessary for test and integration. In addition many of the systems would not be available at test time. Emulation, combined with simulation, has been discussed as one solution to this problem. Its

advantages relative to compilation/assembly and translation have been examined. Although emulation is an attractive alternative, it should not be pursued unless the following conditions are satisfied:

- An easy way of producing microprograms is needed. This requires a high level language. Research has demonstrated that it is practical to use high level languages for producing efficient microcode.[9]
- The host should have a writable control storage for allowing multiple emulator programs, representing a variety of target machines, to be utilized.
- Related to the preceding items is the need for a user microprogramming capability. In a study of user microprogrammable systems it was concluded that this capability can improve both the space utilization and execution time performance of a computer system.[10]
- The host should not differ greatly from the target machine in terms of CPU and memory characteristics: word length, number of registers and memory size. Where differences do exist, the host should have the greater capability.

REFERENCES

1. Husson, Samir S., *Microprogramming: Principles and Practices,* Prentice-Hall, Inc. 1970, p. 94.
2. *Design of Microprogrammable Systems,* Scientific Micro Systems, Inc., July 1973, 10 pages.
3. Mallach, Efrem G., "Emulator Architecture," *Computer,* August 1975, pp. 24-32.
4. Jaeger, Robert, "Microprogramming: A General Design Tool," *Computer Design,* August 1974, pp. 150-157.
5. Jones, Louise H., "Survey of Current Work in Microprogramming," *Computer,* August 1975, pp. 33-38.
6. Leventhal, Lance A., "Microprogramming and Simulation: A Review" *Proceedings of the Summer Computer Simulation Conference,* 1977, pp. 103-106.
7. Wachs, R. E. and B. A. Claussen II, "Software First: Our Viking Experience and Continuing Research," *Proceedings of the Summer Computer Simulation Conference,* 1977, pp. 108-111.
8. Ramamoorthy, C. V., "A Survey of the Status of Microprogramming," *Advances in Information Systems Science,* Vol. 5, Plenum Press, 1974, pp. 193-257.
9. Rosin, Robert F., et al., "An Environment for Research in Microprogramming and Emulation," *Communications of the ACM,* August 1972, pp. 748-760.
10. Rossman, George F., et al., "The Technical Significance of User Microprogrammable Systems," National Bureau of Standards Contract No. 4-36045 awarded 19 June 1974, 44 pages.

PRIM system—A framework for emulation-based debugging tools*

by JOEL GOLDBERG, ALVIN COOPERBAND, and LOUIS GALLENSON
University of Southern California Information Sciences Institute
Marina del Rey, California

INTRODUCTION

For some applications the native machine is not the system of choice in which to develop software, as when the target machine is unavailable (because it is still being developed, is obsolete, or is inaccessible) or inconvenient (as when there is minimal target-system support for debugging). In such cases, simulation or emulation may be preferred. Simulation has the advantage of giving the user intimate access to the target machine, usually through a rich debugging package. Typically this richness is achieved at a high development cost for the simulator and at a target-system performance degradation of four or more orders of magnitude. Emulation can offer processing speeds comparable to the target system (even faster, for slow target machines), but typically does not support a rich debugging environment. In designing the PRIM system, we have attempted to retain the best features of both the simulation and emulation approaches while at the same time minimizing their disadvantages. PRIM provides a sharable, uniform framework for running emulations of target machines; within that framework is a rich user interface that supports interactive target-system and emulator debugging. When the user is not engaged in debugging, the target system runs at emulator speeds, but a sophisticated debugging package is available immediately when needed. PRIM was developed within a modern timesharing system so as to provide convenient access, a file system, resource management, and a large set of utilities without the cost of developing yet another operating system.

THE PRIM SYSTEM ARCHITECTURE

The emulation of a target machine under PRIM involves three different system levels: a timesharing system, which runs on a DEC PDP-10; the PRIM framework, which runs at user level on the PDP-10; and target-machine emulation tools controlled by that framework, which run on a sharable MLP-900 microprogrammable processor. (The MLP-900—Multi Lingual Processor—is the prototype of a microprogrammable processor designed by Standard Computer Corporation[1] to follow their IC-7000.) This architecture is shown schematically in Figure 1. A more complete description of the PRIM architecture was presented by the authors at the Tenth Annual Workshop on Microprogramming.[2] The timesharing system hardware and software provide shared access to the MLP-900. The PRIM framework supports interactive users at terminals and provides access to the file system for the emulator. And the emulator maintains a complete target-system environment. The PRIM framework can be used for both emulator development and target program debugging.

The PDP-10 is a large, general purpose computer to which new devices can be connected fairly easily, since the I/O bus is extensible and the multiported memory is external to the processor. At USC Information Sciences Institute (ISI), it runs the TENEX timesharing operating system developed at Bolt, Beranek, and Newman.[3] TENEX is strongly oriented toward the support of interactive computing, serving both local users and remote users connected via the ARPANET. The operating system does not interact directly with the user; rather, it allocates resources, manages the file system, and supports the execution of TENEX processes, each process running in its own paged virtual memory and interacting as appropriate with its own user via a terminal of some kind. To support PRIM, TENEX was extended with software to provide access to the MLP-900 by TENEX processes; the MLP-900 was extended with hardware and software to guarantee the integrity of TENEX even against errant microcode.

The MLP-900 is a large, fast, vertical-word, microprogrammable processor with a writable control memory. The processor consists of an operating engine and a control engine. The operating engine is a 36-bit arithmetic/shift unit with 32 general registers, 16 mask registers, and a 1K internal memory. The control engine is a control unit with interrupt and branch logic, a subroutine-call stack, 128 programmable flip/flops, and 4K of writable control memory. Cycle time is 250 nanoseconds, during which either one or both engines can execute a 32-bit instruction. The MLP-900

* This research was supported by the Defense Advanced Research Projects Agency under Contract No. DAHC15 72 C0308, ARPA Order No. 2223, Program Code No. 3D30 and 3P10. Views and conclusions contained in this paper are those of the authors.

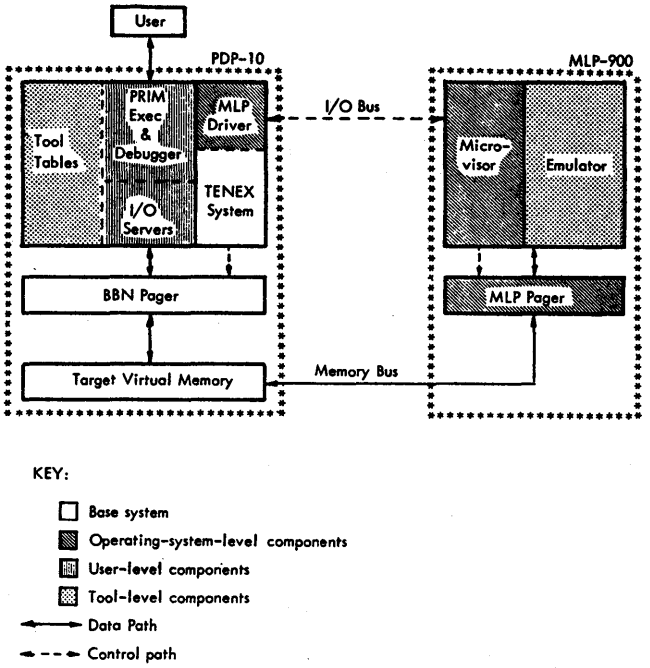


Figure 1—Architecture of PRIM

is interfaced as a peripheral device on the PDP-10 I/O bus with direct access to the PDP-10 memory via one of the four existing memory ports; it has no peripheral devices of its own. The I/O bus interface allows the exchange of control information between the MLP-900 and the PDP-10; via this interface, either processor can interrupt the other. Hardware modifications were required only in the MLP-900; they consisted of the interfaces to the PDP-10 and a supervisor/user state to protect the I/O bus interface, the MLP pager (an address translator in the memory interface that mimics the TENEX pager), most of the MLP-900 interrupt system, and the MLP-900 control memory itself against user microcode.

The software at the system level consists of a small operating system resident in the MLP-900, known as the microvisor, and a new TENEX device driver to shake hands with the microvisor and govern access to the MLP-900 by TENEX processes. The MLP device driver is the sole TENEX operating system module added. It allows a TENEX process to create, run, and control a subordinate MLP process in much the same way it can a subordinate TENEX process. It schedules use of the MLP-900 among contending users and supervises the microvisor. Most of the microvisor is devoted to swapping emulator contexts (control memory and MLP registers) as the driver passes control of the MLP-900 from one user to another. The process of swapping an emulator's context into and back out of the MLP-900 requires about ten milliseconds of microvisor execution and introduces no perceptible delays beyond those ordinarily encountered in a timesharing system. The rest of the microvisor responds to emulator requests for service, manages the MLP pager, and performs other tasks required by the driver in TENEX. The microvisor runs in the privileged supervisor state that allows access to all resources; emulator microcode runs in the user state that protects all the critical resources from modification. PDP-10 memory is not directly addressed by microcode; instead, memory references are to addresses in a virtual memory identical to that of a TENEX process. These virtual addresses are translated to real addresses by the MLP pager, which is controlled (via the microvisor) by the driver in TENEX. A reference to a page not in memory results in a page-fault interrupt into the microvisor, which passes the fault on to the driver and retries the memory operation after the page is fetched by TENEX.

The net effect of the design is a sharable emulation facility in which each emulator runs independent of all others in its own context, accessing its own virtual target memory under control of the PRIM framework that created it. The framework has potential access to all of its emulator's context and target memory and may inspect and/or modify them.

The PRIM system development started in 1972. The initial hardware and software were operational in May 1974 with the ability for multiple (local and remote) users to develop and debug emulator microcode. The complete system became available in March 1976. All software and hardware were developed at ISI. System software took about one man-year. The TENEX MLP-900 device driver represents about 2000 machine instructions; the MLP-900 microvisor occupies about 500 words of control memory.*

THE PRIM FRAMEWORK

The PRIM framework consists of TENEX processes that define and implement the PRIM user command language, create an MLP-900 emulation process and control its execution at the user's behest, and provide I/O service for that emulation process. The I/O service implements a set of primitives that allow the emulator to transfer data to or from the TENEX file system. The emulator invokes these primitives to perform target I/O operations on installed devices after the user has associated them with TENEX files.

An emulator is required to cooperate in the debugging process, although the demands are minimal. Essentially, an emulator is expected to stop cleanly when interrupted by the framework or on the occurrence of a small number of predefined events that it monitors and to report its reason(s)

* It is worth noting that the operating-system level of PRIM was designed, developed, and tested using a TENEX system that, except for maintenance, was in production use continuously. The hardware interfaces were checked out using a dedicated 16K-word memory module and a specially built I/O bus simulator that connected the MLP-900 to the PDP-10 via a high-speed EIA terminal line. A private MLP-900 driver controlled this "smart terminal." After the microvisor was developed and the swapping and paging algorithms validated, the memory module was shared among concurrent MLP-900 users. After validation of the hardware and software interfaces between the MLP-900 and the PDP-10, the I/O bus simulator was removed and the MLP-900 was connected to the PDP-10 I/O bus. An MLP-900 device driver was then added to TENEX and the MLP-900 connected to the full TENEX memory; since most of the driver logic had already been checked out, only a short extension of maintenance periods was required for stand-alone checkout of the new device driver. The change-over to this driver had no effect on the microvisor.

for stopping. When the emulator halts or is interrupted by user intervention, control returns to the user at command level via the framework. Specific commands are available to start (or resume) execution.

The emulator writer must supply the PRIM framework with tables that define the target system architecture and symbols and drive a target system assembler and disassembler. Except for machine and user symbols and target assembly language, the same command interactions apply to every use of an emulator and framework. The framework contains two separate command-processors, known as the exec and the debugger. The exec has a large number of commands of fixed form that are used infrequently. The debugger has a smaller number of commands, but they are of variable form and are used much more frequently. Although both command processors offer automatic command completion and help facilities, each uses a language tailored to its functions. Typically, a user interacts with the exec only briefly when starting and ending a session; during the session he interacts primarily with a target program or the debugger. The exec commands are based on easily remembered keywords that often require several characters to establish uniqueness; the debugger commands are uniquely identified with only a single keystroke.

The PRIM debugger is a target-system-independent interactive program for debugging a PRIM emulator or a target program running on such an emulator. Basically, it permits a user to examine, search, and modify target system locations and to set and clear breakpoints. Target system assembly language and symbols are recognized and generated, expressions of arbitrary complexity are accepted, and arithmetic is performed according to the conventions of the target machine. The debugger commands are listed in Table I.

Representation commands allow the user to designate the form and notation of the debugger's representation of data (FORMAT, MODE, and the three SYMBOL commands). Target system data may be represented as text or instructions or as formatted or unformatted numeric values; numbers may be represented as symbols or as signed or unsigned integers. The user may define multi-field data formats, select user symbol tables loaded earlier via an exec command, and add new symbols to a symbol table or delete existing ones.

Control commands allow the user to execute debugger commands conditionally or indirectly (IF and USE), to continue execution of a target system, either at the system's program counter or at a supplied address (GO), to single-step a target program (SINGLE STEP), to set and clear breakpoints in the target system and enter or edit debugger command "programs" to be executed at break time (BREAK, DEBREAK, and PROGRAM EDIT), and to return control to the PRIM exec (RETURN); privileged commands permit an emulator writer to single-step or set or clear a break in the MLP-900 (MLP STEP and MLP BREAK). Conditional and indirect execution are particularly important when a command is to be executed at a later time in an indeterminate context, such as in a command file or a break-time program. Breakpoints can be set on references (read, write, or execute) to target machine locations or on events that have been predefined by the emulator writer. The most powerful features of breakpointing is the ability for a user to associate with each breakpoint a sequence of debugger commands (the break-time program) to be executed when the break condition is detected and control is passed to the debugger. Coupled with conditional and indirect execution of debugger commands and the ability to continue target system execution, break-time programs offer the user a very powerful facility to trace execution, monitor events, and perform other complex exploratory functions automatically and repeatedly.

TABLE I.—PRIM Debugger Commands

Representation	Control	Display	Miscellaneous
FORMAT	MLP BREAK*	MLP TYPE*	MLP CHANGE*
KILL SYMBOL	MLP STEP*	EQUALS	CLEAR
MODE	BREAK	EVALUATE	COMMENT
NEW SYMBOL	DEBREAK	JUMP HISTORY	SET
OPEN SYMBOL	GO	LOCATE	
	IF	NEXT	
	PROGRAM EDIT	PRIOR	
	RETURN	SAME	
	SINGLE STEP	TYPE	
	USE		

* Commands for emulator developers only.

Display commands allow the user to examine the contents of designated target locations (TYPE, PRIOR, NEXT, SAME, and EQUALS), to search extended areas within the target system according to content (LOCATE), to evaluate arbitrary expressions (EVALUATE), and to review a record of recent target-system control transfers (JUMP HISTORY); a privileged command permits an emulator writer to display MLP-900 control memory symbolically (MLP TYPE). Any displayed location may be modified.

The miscellaneous commands offer convenience features that do not fit in the other categories. They allow the user to change the contents of target system locations without having to display them first (SET and CLEAR) and to annotate transcripts or break-time output (COMMENT); a privileged command permits an emulator writer to compile code changes directly into MLP-900 control memory (MLP CHANGE).

The PRIM exec comprises a diverse collection of commands that are not directly involved in the intimate control of debugging. The exec commands are listed in Table II. One major group of commands is concerned with the establishment of a target machine configuration (INSTALL, SET, and UNINSTALL) and with the mounting of TENEX files on the configured I/O devices (CANCEL, MOUNT, REASSIGN, REWIND, and UNMOUNT). Another group of commands involves the loading of various classes of state information from files (LOAD, TABLES, RESTORE, and SYMBOLS) and with the saving of all or part of the target context on a file for subsequent restoration (SAVE). There are commands that transfer control from the exec to some other state (QUIT, DEBUG, and GO), and commands that alter the state of the PRIM framework (NO, CHANGE, COMMANDS, and ENABLE). PRIM will, on request, maintain a transcript of all or a part of a PRIM session (TRANSCRIPT and CLOSE); such transcripts have been

TABLE II.—PRIM Exec Commands

Configuration	Save/Restore	Miscellaneous
CANCEL	LOAD*	NO*
FILESTATUS	TABLES*	CHANGE
INSTALL	RESTORE	CLOSE
MOUNT	SAVE	COMMANDS
PERIPHERALS	SYMBOLS	DEBUGGER
REASSIGN		ENABLE
REWIND		GO
SET		NEWS
SHOW		QUIT
UNINSTALL		TIME
UNMOUNT		TRANSCRIPT

* Commands for emulator developers only.

found invaluable by users with CRT terminals—and therefore no written record of their work. The remaining commands (FILESTATUS, PERIPHERALS, SHOW, NEWS, and TIME) are information-displaying commands.

The PRIM framework was written in BLISS.[4,5] It occupies about 35K words of PDP-10 memory, including fixed work space, and took somewhat over three man-years to produce.

EMULATOR MODEL

A prototypical PRIM emulator has been developed based on the constraints of the MLP-900 environment, the objectives of the PRIM system, the requirements of the PRIM framework, and the specific interface conventions that framework defines.[6] The environment consists of execute-only microcode residing in control memory, the MLP-900 registers, and a 256K 36-bit (virtual) main memory; the registers and virtual memory together comprise the context into which are mapped the registers and memory of the target machine plus various other regions devoted to required PRIM functions. The mapping is arranged at the convenience of the emulator writer, with accompanying tables describing this mapping to the PRIM framework. The emulator can modify its context in the course of emulation, stop (thereby returning control to the framework), and request I/O services from the framework.

The most fundamental PRIM requirement is for a bit-compatible emulation of the target machine with accurate timing. Beyond that, an emulator must have a control structure that allows it to stop after any cycle and subsequently resume emulation in a manner totally transparent to the target machine. While a single target instruction constitutes the typical cycle, other events, such as interrupts or I/O data transfers, must also be treated as emulator cycles. Changes to the context made by the user during an emulator stop must appropriately affect the target machine on resumption of emulation. In particular, this requires that the emulator have no hidden copies of target state information when it stops.

Target timing in PRIM is virtual. The emulator is required to increment an internal, high-resolution (typically, 50 nanosecond) clock to reflect the passage of target-machine time; there is no fixed relationship among target time, MLP-900 time, PRIM framework time, and real time. Emulated cycles that consume target time (e.g., instruction execution) advance the clock; emulated cycles that nominally occur in parallel with the former (e.g., I/O controller activity) are scheduled for service relative to that internal clock. The result is a small, event-driven, discrete simulation system with target instruction execution being treated as a background task. Figure 2 is a top-level outline of a typical PRIM emulation tool.

Target peripheral devices are included in an emulation. Device control logic, timing, and interaction with the target machine are contained entirely within the emulator. An I/O transfer is performed by the I/O server in the PRIM framework in response to a call by the emulator, which must then monitor the transfer for completion. The transfer is carried on in parallel with emulator execution, and the emulator may have several I/O operations outstanding simultaneously. The freedom given the user to define target system configurations and the protocols for use of the I/O server primitives, together constitute a generalized framework for device emulation.

User commands to inspect and modify the target machine environment are implemented by the PRIM framework via tables associated with each emulator, requiring no direct intervention by the emulator itself. But user commands involving the breakpoint facility do require the cooperation of the emulator. Each breakpoint a user sets is marked by a flag in a portion of the shared context outside the address space of the target machine; these flags are called "metabits." The emulator must monitor the target machine con-

```
initialize emulator ;
FOREVER DO
BEGIN
  IF reason to stop
  THEN BEGIN
    stop ;
    respond to switches and buttons ;
    END;
  IF time to serve next scheduled event
  THEN BEGIN
    remove event from list ;
    execute event routine ;
    END
  ELSE
    IF interrupt pending
    THEN execute interrupt cycle
    ELSE execute instruction cycle ;
END ;
```

Figure 2—PRIM emulator outline

tinually for the occurrence of any breakpoints flagged by these meta bits and, when one occurs, log it and prepare to stop at the end of the current emulation cycle. Interrupting execution at the end of the cycle is consistent with the requirement for stopping cleanly and also avoids a problem that would arise if execution were interrupted in mid-cycle on the occurence of the break condition—that of responding to possible changes made in the context during the break while not reporting the same breakpoint repeatedly.

PRIM defines two classes of breakpoints, known as reference breaks and event breaks. A reference break occurs on a read (data fetch), write, or execute reference to specified target locations. The meta bits for reference breaks are the four high-order bits of each 36-bit context word. As every target location can have its own meta bits, there is no practical limit on the number of simultaneous breakpoints the user may set. An event break occurs whenever a specified target-machine event arises. Each emulator supports its own set of event breaks, but at a minimum includes instruction execution (target machine single-step), jumps or branches, memory stores, interrupts, and I/O transfers. One meta bit is used for each type of event break an emulator monitors. The event breaks are particularly useful for close monitoring of target machine behavior.

To date, there are three supported PRIM emulations: the AN/UYK-20 (a modern 16-bit computer),[7] the Univac U1050 (an ancient byte-addressed decimal machine with a 30-bit instruction word),[8] and the Intel 8080 chip. Both the AN/UYK-20 and the U1050 emulations include a comprehensive set of peripherals. These two emulators each require about 3K words of control memory; the 8080 emulator, which supports only a simple terminal-like device, requires about 1K words of control memory. For the AN/UYK-20, which is a well-documented system of moderate complexity, the emulation of the CPU and the peripherals each required about six man-months. Although the U1050 is of comparable complexity, development of the emulator took somewhat longer because of the unavailability of accurate and up-to-date documentation. The 8080 emulation took only one man-month. All of these emulators require on the order of five microseconds to emulate one target memory cycle. For the AN/UYK-20, the ratio of emulated-to-real time is about 10/1; for the U1050 and the 8080, the ratio is about 1/1. For all emulations, as the I/O traffic increases, the performance ratio increases.

DISCUSSION

Since it became operational, PRIM has been used extensively by the System Design Laboratory (SDL) at the Naval Ocean Systems Center in San Diego, by the Logistics group at Gunter Air Force Station, and by a "smart terminal" project at ISI. SDL is using PRIM to support the design and development of naval systems employing the AN/UYK-20 minicomputer. The Logistics group at Gunter has used PRIM to examine the consequences of replacing existing peripherals with higher-performance ones on a Univac U1050 system. In the process, they discovered an unexpected design limitation in a communications interrupt routine and also uncovered several system bugs for which patches were developed on the spot. An Intel 8080 emulation was used at ISI to debug terminal software that would otherwise have been impossible to observe. The MLP-900 has been used an average 20 hours per month (representing an average of about 40 hours of connect time) by as many as six concurrent users. During this time an average of ten hours of TENEX time per month was used by the PRIM framework. The maximum level of use to date has been 90 hours of MLP-900 time in one month. Ordinarily, a PRIM user is unaware that he is sharing the MLP-900 with other users.

By cleanly and sharply separating the debugging and target-machine emulation tasks, PRIM has been able to avoid most of the disadvantages of simulation and emulation while at the same time combining their advantages. In achieving this sharp separation of function, PRIM established a uniform and systematic structure for the development of emulators. This structure not only minimizes the involvement of the emulator in the debugging process, but also greatly simplifies the task of emulator development as it utilizes a standard package of I/O service routines and provides a convenient control structure suitable for a large family of target-machine emulations. As most of PRIM consists of sharable system-level and user-level code that is common to this potentially large family of target system emulations, a more extensive development effort (with its consequently more sophisticated design) could be justified than would have been appropriate for a single-machine emulation or simulation.

REFERENCES

1. *MLP-900 Multilingual Processor—Principles of Operation,* Standard Computer Corporation, Santa Ana, Calif., 1970.
2. Goldberg, J., A. Cooperband and L. Gallenson, "The PRIM System: An Alternative Architecture for Emulator Development and Use," *MICRO10 Proceedings,* October 1977, pp. 1-6.
3. Bobrow, D. G., J. D. Burch, D. L. Murphy and R. L. Tomlinson, "TENEX, A Paged Time-Sharing System for the PDP-10," *Communications of the ACM,* Vol. 15, No. 3, March 1972, pp. 135-143.
4. Wulf, W. A., D. Russel, A. N. Habermann, C. Geschke, J. Apperson, D. Wile and R. Brender, *BLISS Reference Manual,* Carnegie-Mellon University, 1970.
5. *BLISS-10 Programmer's Reference Manual,* Digital Equipment Corporation document DEC-10-LBRMA-A-D, Maynard, Mass., 1974.
6. *PRIM System: Tool Builder's Manual* (in progress).
7. Gallenson, L., A. Cooperband and J. Goldberg, *PRIM System: AN/UYK 20 User Guide and User Reference Manual,* ISI document TM-77-5, October 1977.
8. Gallenson, L., A. Cooperband and J. Goldberg, *PRIM System: U1050 User Guide and User Reference Manual,* ISI document TM-77-6, October 1977.

A microprogrammed AN/UYK-20(V) emulation*

by DONALD A. DEEL and WALTER A. BURKHARD
University of California
La Jolla, California

INTRODUCTION

In this paper an emulation of the Sperry Univac AN/UYK-20 computer on the Nanodata QM-1[2] is described; the paper is divided into four sections. Short descriptions of both the target machine and the host machine are presented in the first section, each emphasizing those characteristics which most affected the emulation. The second section consists of a description of the basic structure of the emulation. This includes brief explanations of the flow of control through the various parts of the emulation, the principal routines, and the allocations of storage within the QM-1. Also, inaccuracies in the emulation are noted, with an indication of their effect and how they might be remedied. The third section gives a sampling of emulation performance characteristics and in the final section future plans are presented.

The AN/UYK-20 target machine

The AN/UYK-20 is a militarized 16 bit minicomputer. It possesses a large instruction set and good I/O provisions. The following paragraphs form a brief description of the machine.

Memory

Main memory consists of up to 64K words of 16 bits each. Several memory locations are dedicated to specific purposes: locations 110-137 (octal) are for processing interrupts; locations 140-141 (octal) are the I/O command cells, which are used to start up I/O transfers; and locations 200-217 are for external interrupt word storage.

Registers

There are two banks of general purpose registers, with sixteen 16 bit registers in each bank. Only one bank can be in use at one time; which bank is currently in use is determined by a bit in the Status Register #1. This register also indicates which interrupt classes are locked out, what the current condition code is, and what the carry and overflow bits are. The Status Register #2 controls the type of addressing that will occur for certain kinds of memory references. It also holds I/O instruction fault and memory resume interrupt data.

Other registers include a Real Time Clock, a Monitor Clock, and 64 Page Registers. The Real Time Clock is a 32 bit register which counts up, and the Monitor Clock is a 16 bit register which counts down to zero. The Page Registers are used to affect all memory accesses. The upper 6 bits of each 16 bit address coming from the CPU is used to access one of the 64 Page Registers; the six-bit contents of the selected page register is then substituted for the original upper 6 bits of the memory address.

Instruction formats

The AN/UYK-20 has both single and double word instructions. Figure 1 presents the various formats.

I/O provisions

There are sixteen I/O channels; each with dedicated control memories. Figure 2 displays the data formats within each I/O channel control memory.

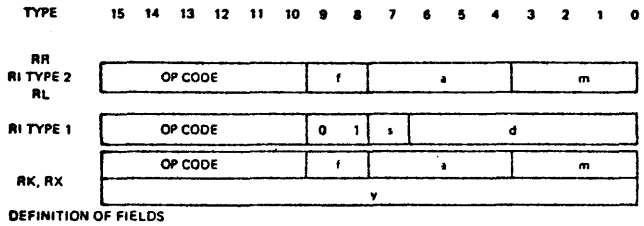
A Control Memory holds all the information required by the I/O channel to perform data transfers. I/O instructions that control the input and output activity on all peripheral channels are executed under an active chain that is associated with each input and each output channel. I/O activity is initiated in the channels by placing the "initiate chain" instruction in the command cells in main memory (locations 140-141) and executing the "I/O command" CPU instruction.

The I/O Channels are capable of performing 8 bit or 16 bit parallel data transfers. 32 bit transfers can be affected by using two channels wired together. Serial I/O channels are available which meet the characteristics specified in EIA standard RS-232C, MIL-STD-188C, or NTDS serial specification.

Interrupts

There are three classes of AN/UYK-20 interrupts. From highest to lowest priority, these are: hardware error interrupts; software interrupts; and I/O interrupts. Within each

* This work supported in part by NOSC Contract No. 66001-77-C-018.



DEFINITION OF FIELDS

OP CODE: CODE SPECIFYING THE OPERATION

f: FORMAT DESIGNATOR
- f = 00 FORMAT RR-REGISTER-REGISTER OR RL-1 FORMAT
- = 01 FORMAT RI-REGISTER-IMMEDIATE MEMORY OR RL-2 FORMAT
- = 10 FORMAT RK-REGISTER-LITERAL CONSTANT OR RL-3 FORMAT
- = 11 FORMAT RX-REGISTER-INDEXED ADDRESS OR CONSTANT OR RL-4 FORMAT

a: GENERAL REGISTER OR SUBFUNCTION DESIGNATOR

m: GENERAL REGISTER OR SUBFUNCTION DESIGNATOR

m: 4-BIT UNSIGNED LITERAL IN RL FORMAT

d: DEVIATION (VALUE)

s: SIGN DESIGNATOR FOR d
0 = POSITIVE
1 = NEGATIVE

y: ADDRESS OR ARITHMETIC CONSTANT

Figure 1—AN/UYK-20 instruction formats

class, there is a priority structure to handle simultaneous interrupts.

Each interrupt class has associated with it several dedicated locations in main memory. These locations are similar to the OLD PSW/NEW PSW memory allocations in the IBM 360. Interrupts are responded to by first storing the program counter, status registers, and Real-Time Clock in several of the dedicated locations, and then re-loading these registers from the remaining dedicated locations.

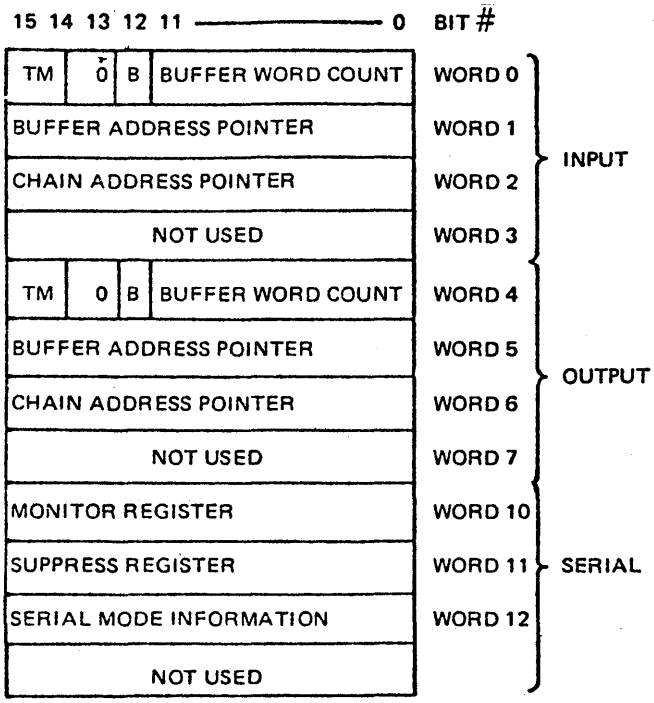


Figure 2—I/O channel control memory

The QM-1 host machine

The Nanodata QM-1 is a machine possessing two levels of microprogrammed control. This is accomplished using three different memories: Nanostore, Control Store, and Main Store. Target Machine (AN/UYK-20) instructions in Main Store are executed by (and defined by) microprograms in Control Store, under vertical control. Microinstructions in Control Store are in turn executed (and defined by) Nanoprograms in Nanostore, under horizontal control. This Control Hierarchy is illustrated in Figure 3.

With appropriate software, including both a micro language definition and systems support functions for the QM-1, the micro-level user can utilize the machine without being concerned with the nano-level. The micro language definition consists of a complete set of nano programs collectively called MULTI,[3] which is well-adapted to writing emulations. The systems support is given by two programs written in the MULTI micro language called TASK[4] and PROD.[5] TASK supports multiple co-resident Control Store tasks in a hierarchical manner. TASK also manages all QM-1 I/O functions for the micro-programmer. PROD (Programmable Run-time Operator's Display) allows the QM-1 operator to use the console to control the emulated target machine. PROD also provides trace/debug capabilities for the emulated hardware system and trace-debug capabilities for the microprogrammer. Since MULTI, TASK, and PROD were used for the AN/UYK-20 emulation, the Nanoprogramming level of the QM-1 will not be discussed in this paper.

The following paragraphs are a brief overview of the QM-1 resources available to the microprogrammer using MULTI.

Control store

Control Store is a fully readable/writable general-purpose 18 bit wide store implemented in 75 ns semiconductor memory. A maximum control store configuration consists of 16K words.

Main store

Main Store is a general-purpose 18 bit core storage consisting of up to 256K words; full cycle time is 800 ns.

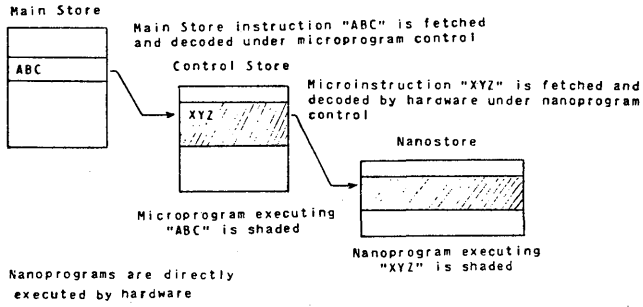


Figure 3—QM control hierarchy

Local store

Local Store is a bank of 32 18 bit registers. Some of these registers have special uses; for instance, the micro program counter and the micro instruction register are kept in Local Store.

Arithmetic-logic unit

The Arithmetic-Logic Unit can be controlled to perform all of the 16 logical (Boolean) operations, as well as common arithmetic operations, upon two 18 bit operands. A 16 bit mode permits the two input operands to be sign-extended from 16 to 18 bits so that the operation of the ALU need not be changed when dealing with 16 bit data values. Also, a decimal control facilitates decimal arithmetic.

Shifter

The 18 bit combinational logic shifter is capable of performing both left and right logical and circular shifts, as well as right arithmetic shifts. Also, double precision (36 bit) left and right logical, arithmetic, and circular shifts can be specified.

THE EMULATION OF THE AN/UYK-20 ON THE QM-1

The emulation of a machine on the QM-1 is very straightforward. It consists of an assembly language program running at the Control Store level of the QM-1. This micro-program explicitly performs all the operations and data manipulations expected of the emulated computer. Thus, object code (for the emulated machine) residing in the Main Store of the QM-1 can be executed in the same manner as it would be in the emulated (target) machine. This micro-program should be considered to be an "action translater;" it converts each of the emulated machine's action requests (the object code instructions) into an equivalent set of QM-1 actions.

In the case of the AN/UYK-20, the micro-program that runs in Control Store has several I/O channels to emulate as well as the Central Processor itself. The micro-program performs the following four operations repeatedly: execute one AN/UYK-20 micro-instruction; service each emulated I/O channel as necessary; update the emulated real-time clocks; emulate any interrupts that might have been caused by the previous three items in this list. Figure 4 illustrates this structure.

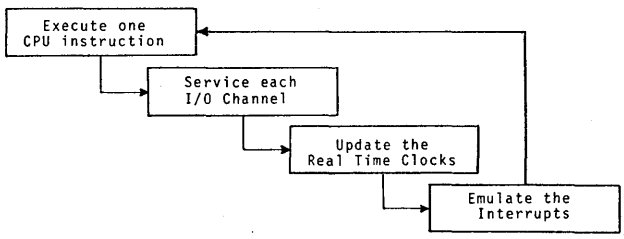


Figure 4—AN/UYK-20 emulation cycle

The following paragraphs are a brief description of the control flow through the various sections of the main emulator loop.

The CPU emulation

The part of the emulation that executes one AN/UYK-20 micro-instruction begins by reading the instruction out of Main Store. The number in the emulated program counter is used as the address. Once the instruction has been obtained, the six-bit opcode field is inspected to determine whether it is a Central Processor command, or an I/O instruction. If it is an I/O instruction, then an error flag is immediately set that tells the interrupts emulation to generate the equivalent of the AN/UYK-20 Central Processor instruction fault interrupt; the flow of control then passes to the interrupts emulation.

For Central Processor commands, it is normal to decode the opcode of the instruction at this point and then execute some micro-routine that is equivalent to the instruction. In the AN/UYK-20, however, it is possible to pre-fetch at least one of the instruction's operands before decoding the opcode. This is done by a micro-routine which first determines the type of the instruction from the two-bit f field. Then, based upon this information, an operand register kept in the Local Store of the QM-1 is loaded with the appropriate data item which always comes either from the currently active AN/UYK-20 general-purpose register bank or from the AN/UYK-20 memory. All of the various memory reference modes are handled in this pre-fetch routine; these modes include direct, indexed, indirect, and cascaded indirect memory references. Besides the operand, the routine also returns the four-bit A and M fields of the instruction in QM-1 Local Store registers; these parameters are used very frequently in processing an instruction after it has been decoded. Also, the address of the operand is put into Local Store if the operand is from memory. This address is used by the "store" instructions.

After the operand is pre-fetched, the instruction is decoded. This is accomplished by using the six-bit opcode and the two-bit modifier field (the f field) together as an index into a table of 256 Control Store locations. Each entry in this table is the Control Store address of a micro-routine which will execute an AN/UYK-20 micro-instruction. Opcode-modifier combinations which are undefined in the AN/UYK-20 all reference the address of a routine that immediately generates an emulated Central Processor instruction fault interrupt.

Immediately before entering a micro-routine, the QM-1 is switched to the 16 bit mode; consequently, the QM-1 ALU conditions can be used directly to set the AN/UYK-20 conditions code. All other portions of the emulation utilize the full 18 bit width of the QM-1 system.

The micro-routines which actually "execute" the instruction generally only consist of a few micro-instructions that

manipulate the QM-1's resources to implement the instruction. Routines that do not affect the AN/UYK-20 conditions code switch the QM-1 back to 18 bit mode and then exit to the next portion of the emulation. Routines that set the conditions code exit to one of two conditions-setting routines (single and double precision) before setting 18 bit mode and going to the next emulation section.

The I/O emulation

Two of the possible sixteen I/O channels of an AN/UYK-20 are presently emulated. This is because two channels are sufficient to handle the peripheral devices available to the emulation: a card reader, a lineprinter, and a CRT console. Additional channels can easily be added for more devices.

The emulation maintains a special flag which indicates whether any I/O activity is taking place. If this flag is not on, then the I/O emulation is exited immediately. This mechanism allows the CPU emulation to run as rapidly as is possible. Otherwise, the channels are emulated serially from highest to lowest priority.

Each emulated channel has a Control Store word of flag bits associated with it that reflect the current status of the channel. AN/UYK-20 channels have both input and output parts that are capable of executing I/O instructions or performing data transfers at the same time. When a channel is being emulated, the micro-program checks both parts to see if they are executing some I/O chaining program that is either active or waiting. Active chaining programs, or chains, are serviced by executing one I/O chaining command from memory. Waiting chains are waiting for some I/O transfer to complete, and are serviced by setting the chain "Active" when the transfer is finished. When an input or output portion of a channel is neither active nor waiting, it is dormant and requires no servicing.

AN/UYK-20 Channels have Control Memories; in the emulation, these are kept in Control Store. The Control Memory of a channel holds information needed for both the input and output portions such as: a buffer address, a word count, and a chain address pointer, which holds the address of the next I/O command to be executed if that portion of the channel is active. The entire I/O interpreter is structured as a subroutine so that each channel with an active chaining program could easily be serviced. This routine begins by fetching an I/O command from memory. The chain address pointer in the channel's control memory is used as the address. After the fetch, the chain address pointer is incremented to point at the next instruction. The opcode is then inspected; if it is not a valid I/O Command, control immediately passes to a micro-routine that generates an emulated I/O Instruction fault interrupt. Otherwise, the instruction's operand is pre-fetched. The major difference between the I/O pre-fetch routine and the Central Processor pre-fetch routine is that there are no indirect references in I/O Commands. After the pre-fetch is done, the instruction is decoded via a table of micro-routine addresses. As in the Central Processor emulation, undefined opcodes result in entries in the table that all point to one routine that generates an emulated I/O Instruction fault interrupt. Each micro-routine that actually executes the I/O command ends by returning to the routine that called the I/O command interpreter subroutine.

Emulation of the real-time clocks

The emulated Real-Time Clocks do not run in true real time; they are run at the same speed as the rest of the emulation. The actual execution times of Central processor instructions and I/O commands are well-known; these times make up the entries in a Control Store table. Just after an construction has been executed, this table is referenced, and the entry corresponding to the last instruction is added to the emulated Real-Time Clock whenever it is enabled. Similarly, the emulated Monitor Clock is decremented by the entry if the Monitor Clock is enabled. When the 16 bit Monitor Clock goes to zero, the emulated Monitor Clock interrupt is generated. When the lower 16 bits of the 32 bit Real-Time Clock overflows, the emulated Real Time Clock interrupt is generated. Running the Real-Time Clocks in this "relative" time keeps events occurring in the emulation in the same relative order that they would have had in the AN/UYK-20 itself.

Emulation of the interrupts

All interrupts are emulated in a single routine so that the proper priority scheme can be maintained. The Central Processor, I/O, and Real-Time Clock emulations generate interrupts by setting the appropriate bits in a QM-1 Local Store register.

The interrupts emulation proceeds by checking each of the enabled AN/UYK-20 interrupt classes from lowest to highest priority. Within each enabled class, each of the applicable emulation interrupt flags are checked, from highest to lowest priority; when an interrupt flag is found on, it is reset and the interrupt is immediately emulated. No further checking of interrupt flags is done for that class. This process of checking for interrupt flags is then repeated for the next higher priority enabled class of interrupt.

Interrupts are emulated by implementing the Old PSW/New PSW interrupt structure of the AN/UYK-20. When an interrupt must be honored, the emulation stores the AN/UYK-20 program counter, status register #1, status register #2, and the Real-Time Clock in Main Store locations dedicated for that purpose. These registers are then loaded from an adjacent set of Main Store locations that contain what is commonly called the "New PSW"; this causes the desired interrupt response, since the Program Counter has been changed. The emulation also calculates an interrupt entrance address index for the class and type of interrupt being honored and adds it to the new program counter.

Simultaneous interrupts of different AN/UYK-20 classes result in Old PSW/New PSW switching in order from lowest to highest priority; then the individual interrupts are serviced in order from highest to lowest priority by AN/UYK-20 programs in Main Store.

Inaccuracies in the emulation

The emulation of the AN/UYK-20 has the following inaccuracies:

The mapping of the upper 6 bits of AN/UYK-20 memory addresses through the 64 Page Registers does not occur. The Page Registers themselves exist and can be loaded and read up normally. No real difficulty is foreseen in correcting this; the nanoprograms have been written but not implemented or tested.

The nondestructive read-out (NDRO) memory is not emulated. There are no plans to correct this, since the AN/UYK-20 Bootstrap programs can reside in Control Store. Thus, the NDRO Mode Designator bit in Status Register #1 has no effect whatsoever.

These are the only known discrepancies in the emulation of the basic AN/UYK-20 at this time. The I/O emulation is complete where the following peripherals are concerned: the card reader, lineprinter, and CRT console.

PERFORMANCE

The AN/UYK-20 emulation is not large; it occupies less than 2.5K words of 18 bit Control Store. Additionally, TASK and PROD occupy approximately 2K words of 18 bit Control Store and 4K words of 18 bit main store.

One of the primary concerns with an emulator is its runtime efficiency. With no I/O activity in progress, the AN/UYK-20 emulation executes CPU instructions with an average slowdown factor of approximately 25. This number was determined by first finding the total time required by the QM-1 to execute an entire emulator cycle for several AN/UYK-20 instructions. These times were then compared to the AN/UYK-20 execution times for the same instructions.

Representative times for typical AN/UYK-20 instructions under different instruction formats are presented in Appendix A. It is assumed that no I/O activity is taking place and that no interrupts are pending. Each instruction time is broken down into the times taken to go through each section of the emulation; each time represents the worst case time for that section. The time units are QM-1 T-periods, which are 80 nanoseconds each.

From these examples, it should be clear that there is an emulation overhead of at least 269 T-periods for every AN/UYK-20 CPU instruction executed. This is incurred by the code sections that fetch instructions, decode instructions, check for I/O activity, check for interrupts and check for emulator single-step mode. The only sections that vary in execution time are those which pre-fetch operands, execute instructions, and set conditions (some instructions do not set conditions). It is this emulation overhead which gives rise to the rather severe slowdown factor of 50.5 in the case of the RR format Load instruction. The overhead time of 269 T-periods constitutes more than half of the execution cycle time of 474 T-periods. This is not true in the case of RX format instructions and instructions which take longer to execute on the AN/UYK-20. This can be seen in the RX format Load and the two Load PSW instructions, which have slowdown factors of 19.3, 13.3 and 11.6, respectively. The last instruction timing breakdown is given as an example of an instruction where the emulation overhead is not a dominate factor.

The overhead incurred by checking for I/O Activity, Interrupts, and Single-Step Mode can be reduced by an order of magnitude. This will result in a 15 percent reduction in the time required to execute an AN/UYK-20 instruction.

FUTURE PLANS

Currently, the AN/UYK-20 emulation on the QM-1 correctly executes the entire Univac AN/UYK-20 Processor 1 Diagnostic test program.[6] This thoroughly tests the CPU functions. Also, several I/O tests have been run in which the emulation has had all the connected peripherals busy while simultaneously executing normal CPU instructions.

Currently, the emulation can write and read Main Store images to and from the QM-1 tape drives; the Processor 1 Diagnostic is loaded into memory for execution in this manner. Adding true bootstrap ability will allow other large programs to be easily loaded. This will entail emulating another AN/UYK-20 I/O Channel for the tape drive and putting the appropriate AN/UYK-20 bootstrap code in Control Store.

One of the first "larger" programs to be run on the emulation will be the AN/UYK-20 Level II Operating System. This will allow the QM-1 to be used as a complete, standalone AN/UYK-20 system. It will also require the addition of an AN/UYK-20 disk I/O Channel to the emulation.

The completed emulation can be used for software development. It is well suited to this task, since an operator can exercise much greater control over the emulation than over a hardware AN/UYK-20.

REFERENCES

1. *Data Processing Set AN/UYK-20,* Volume 1—Operation and Maintenance, Sperry Univac, Univac PX 10675-1-1, September, 1974.
2. *QM-1 Hardware Level User's Manual,* Nanodata Corporation, Williamsville, New York, Second edition, 1974.
3. "MULTI Multiprogramming Support System," Nanodata Corporation, August, 1973.
4. "TASK Control Program," Nanodata Corporation, November, 1976.
5. "Programmable Run-Time Operator Display PROD User's Guide," Nanodata Corporation, 1976.
6. *Data Processing Set AN/UYK-20,* Volume 5—Diagnostic Program Listing, Sperry Univac, Univac PX 10675-5-3, April, 1976.

APPENDIX A—TYPICAL AN/UYK-20 EMULATION TIMINGS

AN/UYK-20 opcode 01: LOAD (RR format)

T-periods	
108	Instruction Fetch
86	Operand Pre-fetch (RR format)
63	Instruction Decode and Update RTC

15	Instruction Execution
104	Set Conditions Code
28	Check for I/O Activity
55	Check for Interrupts
15	Check for Emulator Single-Step Mode

Total = 474 T=periods=37.9μs
Actual AN/UYK-20 execution time=0.75μs
Slowdown factor=50.5

AN/UYK-20 opcode 01: LOAD (RX format)

T-periods	
108	Instruction Fetch
155	Operand Pre-fetch (RX format with no indexing)
63	Instruction Decode and Update RTC
15	Instruction Execution
104	Set Conditions Code
28	Check for I/O Activity
55	Check for Interrupts
15	Check for Emulator Single-Step Mode

Total = 543 T-periods=43.4μs
Actual AN/UYK-20 execution time=2.25μs
Slowdown factor=19.3

Note: If indexing is done in the Operand Pre-fetch, the total time goes up by 92 T-periods; this makes the total time=635 T-periods=50.8μs and the slowdown factor becomes 22.6.

AN/UYK-20 opcode 07: Load OSW (RI format)

T-periods	
108	Instruction Fetch
110	Operand Pre-fetch (RI format)
63	Instruction Decode and Update RTC
120	Instruction Execution
0	No conditions are set
28	Check for I/O Activity
55	Check for Interrupts
15	Check for Emulator Single-Step Mode

Total = 499 T-periods=39.9μs
Actual AN/UYK-20 execution time=3.0μs
Slowdown factor=13.3

AN/UYK-20 opcode 07: Load PSW (RX format)

T-periods	
108	Instruction Fetch
155	Operand Pre-fetch (RX format with no indexing)
63	Instruction Decode and Update RTC
120	Instruction Execution
0	No conditions are set
28	Check for I/O Activity
55	Check for Interrupts
15	Check for Emulator Single-Step Mode

Total = 544 T-periods=43.5μs
Actual AN/UYK-20 execution time=3.75μs
Slowdown factor=11.6

Note: If indexing is done in the Operand Pre-fetch, the Slowdown factor becomes 13.6.

AN/UYK-20 opcode 17: Circular Left Double Shift (RK format)

T-periods	
108	Instruction Fetch
123	Operand Pre-fetch (RK format with no indexing)
63	Instruction Decode and Update RTC
748	Instruction Execution
104	Set Conditions Code
28	Check for I/O Activity
55	Check for Interrupts
15	Check for Emulator Single-Step Mode

Total = 1244 T-periods=99.5μs
Actual AN/UYK-20 execution time=3.0μs
Slowdown factor=33.2

Design considerations in a multiprocessor computer for continuous system simulation

by E. O. GILBERT
Applied Dynamics International
Ann Arbor, Michigan

and

R. M. HOWE
University of Michigan
Ann Arbor, Michigan

INTRODUCTION

The current importance of computer simulation of dynamic systems is well-known. Such simulation is useful in system analysis, design, and testing, as well as in the training of human operators. Whenever a computer simulation involves interaction with actual system hardware (including humans) the computation must by definition take place in real-time; i.e., the computer must solve the simulation equations at the same speed with which the actual dynamic system operates. Real-time simulation of dynamic systems plays an important role in many areas of technology, including aerospace, process control, automotive, and other applications. In the simulation of flight vehicles such as aircraft, helicopters, and missiles the computational requirements associated with real-time simulation can become especially severe due to the complexity of the equations to be solved and the high frequencies present in the dynamic behavior of the solutions.

In the early 1950's, when real-time simulation of flight vehicles began to be achieved on a widespread basis, only analog computers possessed the necessary dynamic capability to accomplish the simulation. Over the ensuing two decades, as digital computers have increased dramatically in speed and computing power, more and more of the real-time simulation of flight vehicles (and other dynamic systems) has been taken over by digital computers. For example, today even general purpose digital minicomputers have the capability of real-time simulation of many aircraft.

However, there continue to be many flight-vehicle systems which currently are impossible or uneconomical to simulate adequately in real-time with existing general-purpose digital computers, e.g., small high-performance missiles, high performance aircraft which include flight-control system dynamics, helicopters, and jet engines. In many of these cases the simulation is being performed successfully with hybrid (i.e., combined analog/digital) computing systems. However, because of the potential advantages which all-digital simulation can have over hybrid simulation, e.g., better reliability, easier programming, lower cost and operation from a stored program, there is ever increasing pressure in favor of all-digital simulation. The purpose of this paper is to describe a multiprocessor digital computer with architecture and hardware designed specifically for efficient high-speed simulation of continuous dynamic systems.

REQUIREMENTS FOR REAL-TIME SIMULATION

There are a number of requirements associated with real-time simulation of continuous dynamic systems which should be understood to appreciate the problems peculiar to such simulation. First of all, because of the basic origin of the requirement, a real-time simulation will involve interaction with the real world external to the computer, usually on a more or less continual basis. For example, a computer used for simulation in flight training or for man-in-the-loop engineering studies will receive continuous inputs from the cockpit controls being activated by the pilot and will need continuously to furnish outputs to drive the various cockpit displays. When portions of a large simulation are implemented by using the actual system hardware with the remaining simulation accomplished on the computer, again the hardware furnishes continuous inputs to and requires continuous outputs from the computer. Of course, in the case of digital simulation continuous data will be approximated by discrete time series, but in any case the data-word rate for each input and output variable must be fast enough to avoid compromising the fidelity of the simulation. Thus real-time simulation often involves high input/output data rates. In typical aerospace simulations these rates may exceed several hundred data words per second for each of 100 channels or more.

In cases where the speed of a digital computer used for real-time simulation is being pushed in order to achieve satisfactory dynamic accuracy, it almost certainly follows that a fixed time step for numerical integration must be used.

This also simplifies greatly the A to D (analog-to-digital) and D to A (digital-to-analog) conversion problem when the computer is interfaced to external analog channels.

Many simulations of continuous dynamic systems involve evaluation during each integration step of numerous functions of many variables. For example, the aerodynamic coefficients used for flight simulation can involve several dozen functions of two, three, four, or more variables with a correspondingly large data base required for function storage. Such simulations also require numerous coordinate transformations during each integration frame, i.e., the transformation of vector components from one spacial coordinate system to another.

Another characteristic of many continuous system simulations is the existence of a wide range of dynamic frequencies inherent in the system. For example, a typical air frame simulation might exhibit a phugoid longitudinal mode at 0.01 hertz and at the same time have a short period pitching mode of 1 hertz, a structural vibration mode of 10 hertz, and a control-surface actuator mode of 25 hertz. The integration frame rate must be high enough to handle the highest mode frequency with adequate accuracy. This means that the integration frame rate is extremely high (i.e., step size is extremely small) with respect to the mode corresponding to the lowest frequency. This can lead to errors caused by roundoff if care is not taken. Multiple integration frame rates can also be employed, but most users prefer to avoid that complexity.

Finally, typical continuous system simulations often involve nonlinear functions with discontinuities, especially in the modeling of control systems. Typical examples include linear functions with limiting, relay (on-off) controls, deadzone, hysterisis, etc. Digital simulation of such functions, especially when employing fixed integration frame rates as normally used for real-time simulation, can result in very sizable dynamic errors unless very small integration steps are utilized.

To summarize, real-time digital simulation of continuous dynamic systems, especially flight vehicles, is characterized by many input-output channels with high data rates, numerous multivariable functions requiring large data bases and many high-speed repetitive-type calculations, coordinate transformations which also tend to be repetitive in nature, a very large spread in dynamic frequency, and many nonlinearities with discontinuities. The computer described in this paper has been designed for efficient processing of simulations having these characteristics.

NUMERICAL CONSIDERATIONS

An important parameter affecting both cost and performance in the design of any digital processor is the word size used for arithmetic operations and memory. As stated in the previous section, much of the computational load in dynamic simulation of continuous systems, especially flight vehicles, involves evaluation of multivariable functions. Usually the data involved in the function tables is known with only limited precision, so that 16 bits is more than an adequate word size to represent the data points. Furthermore, with careful scaling techniques the evaluation of functions using table lookup and linear interpolation can be performed with a 16 bit arithmetic processor without any deterioration from 16 bit precision.[1] Under these circumstances it would in fact be inefficient to use a word size significantly larger than 16 bits.

In the case of coordinate transformations the scaling of the required trigonometric functions is such that 16 bit precision can be maintained. Additional algebraic computations are also required in general to compute the time derivatives of the state variables in typical simulations, but in most cases these do not include small differences of large quantities which might reduce the computational precision. Division can be accomplished by generation of a reciprocal function followed by a multiplication. Simple scaling algorithms insure that precision is maintained even when the divisor varies over a large range. Expanded use of function generation instead of algebraic computation in many other formulas further improves the scaling.[2]

Thus for an accuracy range considered more than adequate for continuous system simulation, e.g., 0.01 percent to 0.1 percent, a convincing case can be made for using a fixed point 16 bit computation for the state-variable derivatives. Not only is the 16 bit precision adequate, but also the maximum range of state-variables is either known or can itself be computed, so that efficient scaling to take full advantage of the 16 bit precision is straightforward. The use of true roundoff as opposed to roundup or rounddown in the arithmetic processor prevents bias which might cause error buildup.

However, in the arithmetic involved in the mechanization of numerical integration formulas it is easily seen that 16 bits can represent quite inadequate precision. In particular this is true when a simulation exhibits a very wide range in dynamic frequencies, which can often occur in flight simulation and other examples as noted in the previous section. The problem can be understood by considering a simple integration formula. Assume we wish to solve the equation

$$\frac{dx}{dt} = Kf(x,y,\ldots) \tag{3.1}$$

where $x,y,\ldots$ are state variables in the simulation. The constant K is chosen so that the function f ranges up to unity in magnitude. For simple Euler integration we use the following formula:

$$x_{n+1} = x_n + hKf(x_n, y_n, \ldots) \tag{3.2}$$

where h is the numerical integration step size and $x_n = x(nh)$, $y_n = y(nh)$, etc. Equations similar to (3.2) will be repeated for each of the problem state variables. As a result there will be a maximum scaling constant, K_{max}, associated with one of the equations and a minimum scaling constant, K_{min}, associated with another of the equations. In the flight-equation example of the previous section, $K_{max}/K_{min} = 25/.01 = 2500$. To be conservative, assume

$$\frac{K_{max}}{K_{min}} = 10^4. \tag{3.3}$$

Now the maximum allowable numerical integration step size, h, will depend on K_{max}. In particular, the modal time constant associated with the equation $\dot{x}=K_{max}\, f \cong K_{max}\, x$, assuming $f(x,y,\ldots)$ can be represented quasilinearly by the simple function x, will be $1/K_{max}$. Let's further assume that we take 10^3 integration steps per time constant, $1/K_{max}$. This would yield good dynamic accuracy (~0.1 percent) using Euler integration. To be sure, higher order integration formulas would allow much larger step sizes for comparable accuracy. However, even in that case one would like to preserve the option of using the smaller step size without getting into roundoff problems. Thus we assume that

$$h=\frac{1}{K_{max}}\times 10^{-3}. \tag{3.4}$$

Now let us examine Eq. (3.2) for the case where $K=K_{min}$. Then we can write

$$x_{n+1}=x_n+hK_{min}\, f \tag{3.5}$$

or from Eqs. (3.3) and (3.4)

$$x_{n+1}=x_n+\frac{K_{min}}{K_{max}}\times 10^{-3} f \tag{3.6}$$

$$=x_n+10^{-7} f \approx x_n+f\times 2^{-23}. \tag{3.7}$$

If the function f is computed to 16 bit precision, and we wish the increment $f\times 2^{-23}$ to be added to x_n with 16 bit precision, then clearly x_n must be represented to at least 23+16=39 bits of precision. Certainly 32 bits would be inadequate and 48 bits contains a conservative safety factor. This is the rationale behind using 48 bits for the numerical integration processor even though only 16 bits is used in the computation of the derivative function f. The above argument also explains why many 32 bit floating-point computations which effectively realize only 21 to 24 bits of mantissa precision must actually be mechanized in double precision for simulations involving "stiff" systems (large ratios for K_{max}/K_{min}).

In order to confirm the effectiveness of 16 bits for the computation of state variable derivatives and 48 bits for numerical integration, a number of typical systems have been simulated on a general purpose computer with a program using the word lengths as proposed above, along with true roundoff. These simulations, including nonlinear airframe dynamic equations, have demonstrated the validity of the approach.

TECHNOLOGY CONSIDERATIONS

As we have seen, many continuous system simulation applications impose severe requirements in speed, real-time operation and economics. We also have observed that 16 bit fixed point computation is adequate in many important simulation applications. A significant exception is implementation of integration algorithms, where a 48 bit word length is desirable to accumulate small increments accurately. Reflecting this application insight, a specialized digital computer for continuous system simulation should maximize, in some reasonable sense, its performance to cost ratio by matching real-time speed and numerical requirements to the currently available technology. This kind of optimization is reflected in the choice of memory, the logic family, the system architecture and a software approach.

Factors in this kind of optimization include the following:

(1) large data memory for function data and solution storage (up to 256K words or more),
(2) high data rates to and from data memory,
(3) fast 16 bit addition and multiplication rates for irregularly structured computations,
(4) flexible and fast accumulation of 48 bit operands,
(5) functional parallelism and pipelining of different aspects of the computational tasks,
(6) architectural flexibility to grow capability with minimal interaction on existing hardware and software, and
(7) very high speed analog to digital, digital to analog and digital to digital data transfers.

Requirements (1) and (2), at first glance, are incompatible in a cost to performance optimization, since low cost memory technology (cost per bit) emphasizes moderate speeds. Thus the currently available 4K×1 and 16K×1 MOS RAMS provide the lowest cost memory but with cycle times from 200 to 500 nanoseconds. This compares with the much faster 1K bipolar RAMS which offer 50 nanosecond cycle time but have much higher cost, increased power requirements and less packaging density. This dilemma is avoided by using the slower MOS RAMS in a highly interleaved configuration. This strategy requires data organization that allows overlapped accesses. Since the continuous system simulation application[1] permits this kind of data organization, the architecture to be described uses an inter-leaved memory to realize both economic and speed objectives.

Requirements (3) and (4) can be realized by parallelism or by very fast addition and multiplication hardware. One extreme approach to parallelism would be to use hundreds of inherently slow microprocessors, each implementing addition and multiplication routines. This scheme imposes serious architectural constraints on data management and hardware interconnection. An alternative is to employ the very high speed-to-cost performance of emitter coupled logic (ECL). With ECL technology, a 17 bit by 17 bit product can be obtained in less than 50 nanoseconds. Also, the ECL family permits very high speed data rates between registers and between blocks of processing hardware. For these reasons, ECL was selected.

Requirement (5) just emphasizes the fact that memory and arithmetic speeds have practical limits and, at some point, additional speed improvements must come from parallelism and pipelining. These techniques can give striking performance improvements, but with penalties in increased cost, increased hardware complexity, difficulty in data transmission, and severe programming and software constraints. These penalties become overbearing in the design of a general purpose digital computer. However, for an application-centered digital computer, which limits scope, the penalties

are more easily controlled and parallelism and pipelining offer intriguing possibilities.

In particular, we can realize significant performance advantages by designing each processor for a given function. Each processor of the multiprocessor computer is then different and suited to a class of operations. An important fringe benefit of this approach is ease in meeting the architectural flexibility of Requirement (6). In fact, the numerical integration processor, which will be described, is being added to existing systems with no difficulty.

Requirement (7) is faced in interfacing the simulation computer to real-time hardware. To be compatible with a high speed digital architecture, both the analog to digital and digital to analog converters must be parallel. Fortunately, fully parallel analog to digital conversion is now feasible because of the improved performance and decreased cost of modular analog to digital converters. Digital to digital transfer is now encountered frequently as digital computers become an integral part of real-time hardware. The net effect of these real-time data conversion and transfer requirements is a special interfacing approach to the digital computer hardware. This implies simple, fast, burst-oriented transfer of data blocks to and from the data conversion equipment.

COMPUTER ARCHITECTURE OF THE AD-10

Figure 1 is a block diagram showing the architectural features of the special digital computer suited to the continuous system simulation task. This system, designated the AD-10 and manufactured by Applied Dynamics International, will be described in the context of the major system elements shown in Figure 1. The purpose is to show how an application-architecture viewpoint can lead to truly significant performance improvements.

Data, address, control and status multibus

The data, address, control and status multibus is a parallel ECL bus composed of 16 data lines, 20 address lines and 12 control and status lines. This high speed multibus supports 20 data/address bus transactions per microsecond. These bus transactions pass data to and from data memory, five functional processors, the real-time input-output channel and the host digital computer.

The transactions, as well as all memory and processor functions, are synchronously controlled from a master 40 MHz clock. This synchronous approach maximizes speed by avoiding handshaking delays characteristic of asynchronous control.

Host interface control

All functions of the system are under control of a host computer interfaced to the multibus through the host interface control (HIC). Through the HIC, the host computer loads the program memory resident in each functional processor, loads data into the interleaved data memory, supervises run/halt control of the system, and performs other control, status interrupt and diagnostic functions. The host may be a relatively large or a moderately sized Digital Equipment PDP-11 system. In addition to control functions, the host provides software support for program generation, translators, file management and debugging and diagnostic aids.

Except for these set up and control functions, real-time continuous system simulation is achieved through coordinated programs stored in the five functional processors. These programs access required data from data memory and control the input-output channel. The data, address, control and status multibus, under control of these processors, serve as the data communication path between the processors and data memory.

Interleaved data memory

The interleaved data memory (DM) holds the large data tables for multivariate function generation, solution storage and general data storage. Data is organized in DM so that data bandwidth is at rates up to the 20 MHz bus transaction limit. With up to 64 interleaved 4K pages of 16 bit words (plus parity), full memory bandwidth can be realized in many applications. Addresses for data are generated by the memory address processor (MAP). Memory size is expandable to 1024K words by using 16K RAMS instead of 4K RAMS.

Functional processors

The functional processors, COP (control processor), MAP (memory address processor), DEP (decision processor), ARP (arithmetic processor) and NIP (numerical integration processor), work in an overlapped and coordinated manner on different aspects of the simulation task. All these processors have certain common features, which will be discussed.

The functional processors synchronously execute instructions at a 10 MHz rate. Each processor contains its own program counter and program memory. These program memories, which are loaded through the HIC, are 50 nanosecond, 1 K word, bipolar writable control stores. The word length in each processor is matched to the needs of that processor. In total, these word lengths add to 272 bits. These processor instruction words, fetched with look ahead, execute in 100 nanoseconds and are responsible for the parallel computing power and speed of the system.

The advantages of this distributed processor architecture are significant. First, the relatively independent character of each processor minimizes design and construction problems implicit in 100 nanosecond execution times. Second, the structure of each processor is determined by a few functional requirements. This is a great simplification over the complex job of designing one processor to do all things. Third, the flexible parallel structure allows the system to be tailored to different application without important impact on other ele-

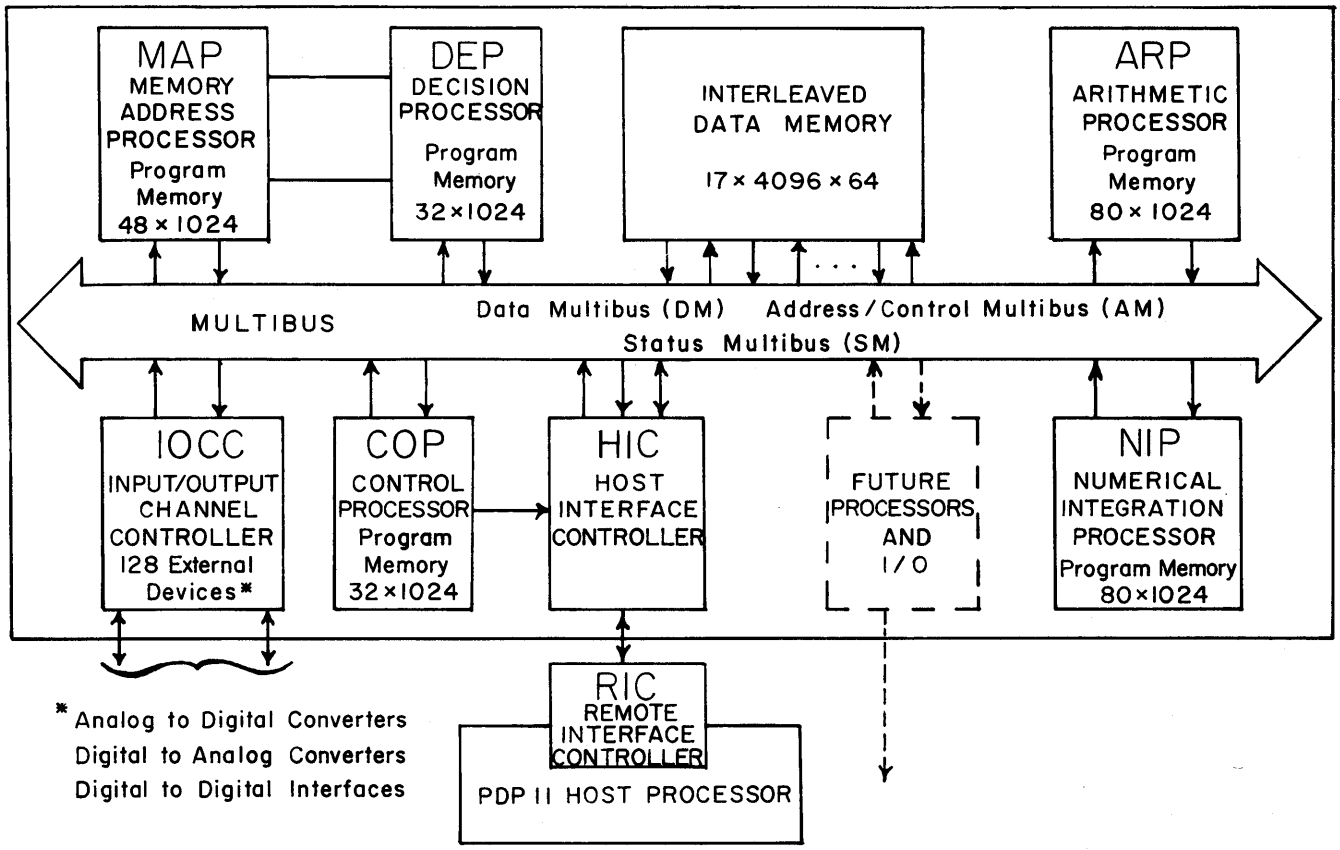


Figure 1—AD-10 block diagram

ments of the system. Fourth, the sequentially coordinated operation of the different processors minimizes programming difficulties inherent in parallel processing. This sequential orientation is consistent with usual programming concepts and avoids the very difficult task assignment and data communication problems which characterize other parallel architectures.

The different functional processors and their roles in continuous system simulation will be considered now in more detail.

Control processor (COP)

The COP coordinates the activities of the other functional processors and provides the means to request service from the host computer. In particular, COP instructions

(1) provide halt and conditional halt control,
(2) control the active/wait status of all processors,
(3) load processor program counters and status words,
(4) read status and data bits for conditional halts and jumps, and
(5) control the real-time input-output channel.

Programmed halts from the COP allow the host computer to control both single frame and periodic operation. Conditional halts permit error conditions or program conditions to halt the program and request host service.

Each processor has an active and wait state. In the active state the program counter is counting and executing instructions; in the wait state a processor is not executing instructions. Through the active/wait instruction, the COP can conserve program memory in a temporarily unneeded processor.

Through load program counter, jump and conditional jump instructions, the COP provides a minimal looping and subroutine capability. Because of the pipelined character of the interleaved memory and, to a degree, the processors, loops should be relatively large for speed effectiveness. Philosophically, the architecture is most capable in executing blocks of branch-free code. Thus, as an example, a linearly coded binary search is used to determine input location between function breakpoints. This technique fully controls the search time and avoids looping.

Memory address processor (MAP)

The MAP generates physical addresses for the DM from MAP user addresses. The physical address consists of a word address (one location in an interleaved page of 4096

locations) and a page address. User addresses are converted to physical addresses by an alignment network which assures that overlapped memory accesses will not be in time conflict in a computational task.

Decision processor (DEP)

The DEP executes comparison and modification instructions on a dual 128 word register file within the DEP. This structure efficiently implements a binary search in function generation,[1] as well as other decision oriented operations. Because these decision operations are isolated within the DEP and because the DEP is associated with the MAP, the ARP is simplified and addressing is made more efficient.

Arithmetic processor (ARP)

The ARP provides the very high speed 16 bit arithmetic needed for continuous system simulation. Through overlapped move and arithmetic operations and a very high speed 128 word register file, the ARP efficiently implements the function generation and irregularly structured computations characterizing simulation applications. Arithmetic is in 2's complement fractional and integer formats. Provisions

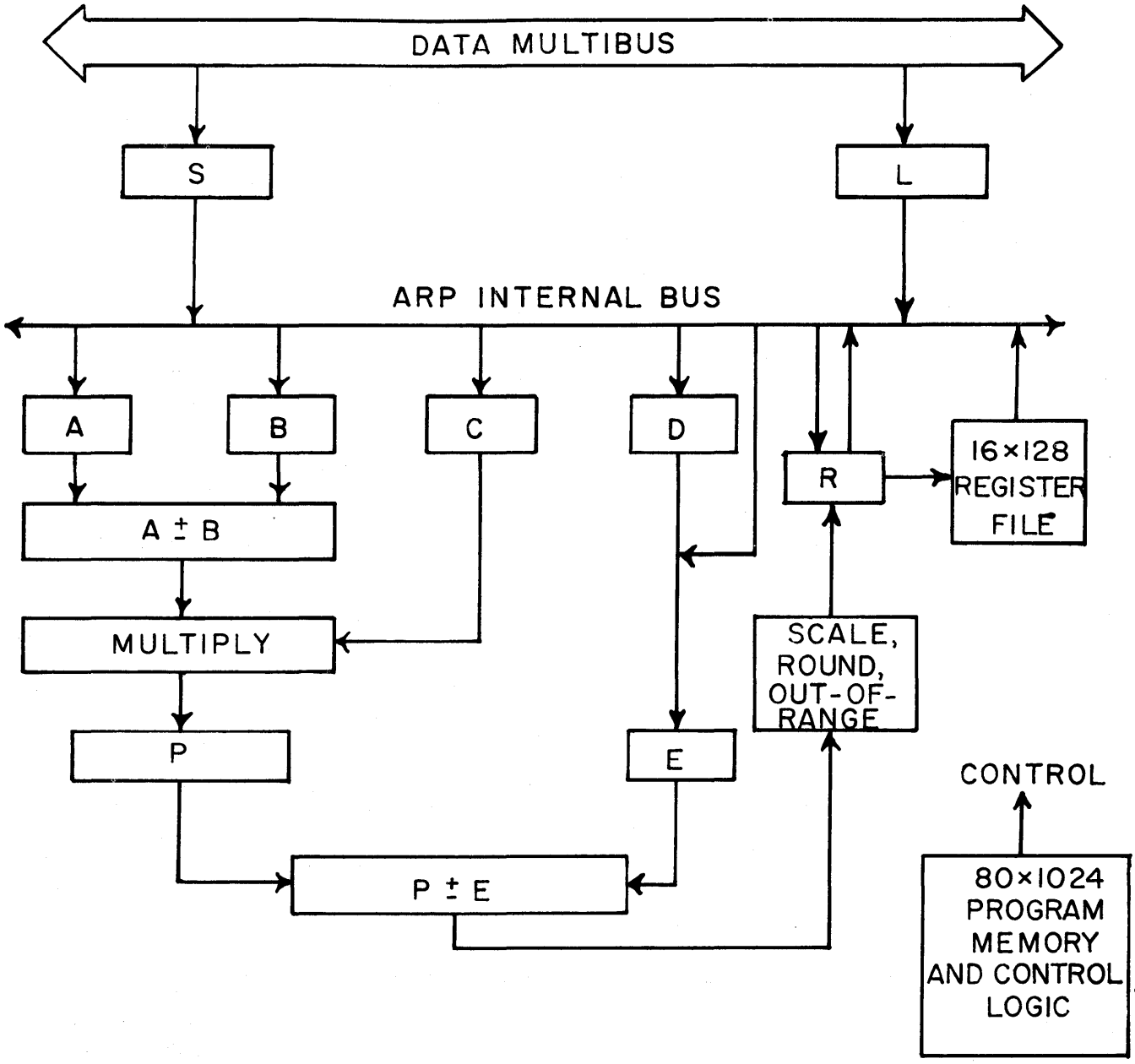


Figure 2—Block diagram of the arithmetic processor

are made for full carrys, scaling and a true out-of-range saturation which avoids modulo 2^{16} wrap-around. These features preserve the effectiveness of the 16 bit format.

The organization of the ARP is shown in Figure 2. Four move subinstructions per instruction set up the operands for the following instruction. This following instruction executes an arithmetic subinstruction of the general form $R=\pm(A\pm B)*C\pm D$ on the operands A, B, C and D, producing the result in 175 nanoseconds. Since all these functions are fully overlapped in a pipelined approach, the result R is computed with a 10 MHz throughput rate. Thus 20 additions and 10 multiplications can be achieved each microsecond.

Numerical integration processor (NIP)

As we outlined earlier, the right hand side of the simulation differential equations is adequately represented by 16 bit arithmetic. The features in the processors described above are effective in implementing these right hand side computations. Many AD-10 systems are currently working in the field performing some or all of these operations. In these situations, the remaining computations and integrations are accomplished in the analog computer subsystem of a hybrid computer system or, in a few cases, in a general purpose digital computer.

The full performance capabilities of the AD-10 for dynamic system simulation are realized when all computations, including integrations, are done within the AD-10. This requires the numerical integration processor (NIP), which is shown in the simplified block diagram of Figure 3. The NIP elements include a 48 bit arithmetic unit, a parallel shifter for accumulating increments to 48 bit words, a 48×1 K register file, buffer stacks and data and control stack subprocessors. The NIP communicates to the other processors through the multibus and is controlled by microprogrammed instructions stored in the 80×1024 program memory.

Data, control and exponent stacks, and indexed addressing to the register file are essential to subroutine implementation of integration algorithms and the effective overlap of integration computation with computation occurring in the other processors. The parallel shift network positions a 16 bit variable according to the sum of four shift factors, E0, E1, E2 and E3. This efficiently implements integration algorithms and allows a "quasi" floating point format for difficult scaling situations which can occur occasionally. The control stack processor permits simple block transfers to other processors and efficiently controls substep logic in Runga-Kutta and other integration formulas. The data stack

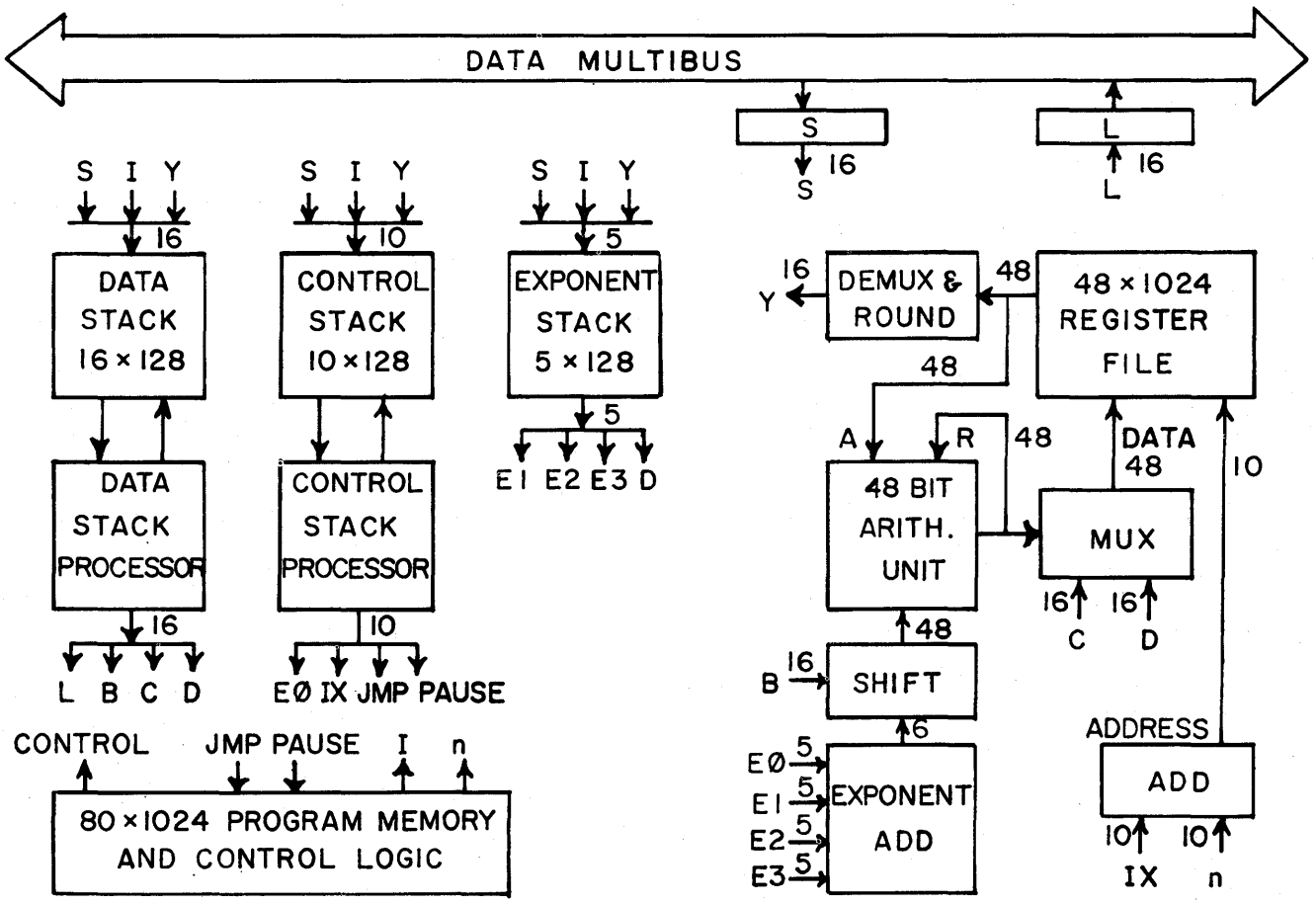


Figure 3—Block diagram of the numerical integration processor

processor is useful for variable step-size algorithms and for implementation of on-off functions, hysteresis, saturation and other simple but common nonlinearities. All these features minimize the interaction of the other processors, maximize overall computing speed and overlap NIP functions with other processor tasks. The general effect is that integration has minimal impact on frame time, which is therefore primarily determined by the other processors.

The instruction fields of the 80 bit NIP instruction are relatively uncoded so NIP programs are microcoded in character. A variety of subprograms are available for implementing a family of integration routines and other functions required in real-time continuous system simulation. The interfacing of these routines to the other processors is very simple, primarily involving data transfers from and to the multibus.

The integration routines are all resident in the NIP so that integration formulas and step size can be changed at run time without any translation requirements in the host. This means that formula selection and step size optimization is a simple process.

SOFTWARE

Since the architecture of the AD-10 is specialized, it is not possible or practical to develop resident software. Instead the host computer and its software systems are used for all program development and interactive control of the AD-10. The Digital Equipment PDP-11 series, operating under RSX-11M, has up to now served in all cases as the host computer for the AD-10, although other host computers can be used.

Available AD-10 software includes interface communication routines, an interactive executive, a cross assembler, a macrofile library, a simulator, and hardware diagnostics. In addition, a high level interactive simulation language is under development.

The interface communication routines are a set of subroutines which perform all the interface control functions between the host PDP-11 and the AD-10. These routines are callable from either Fortran or assembly language programs. They are used by the interactive executive and are also available for user application programs.

The interactive executive (ADX) is an interactive program for loading, running and debugging AD-10 programs. ADX accepts commands from a user at a terminal or from a command file. ADX allows the user to start and stop the AD-10, step its program through specified instruction cycles, and display and modify registers, data memory and program memory. The file structure of RSX-11 is available to ADX to load files created by the cross assembler or to save specified sections of AD-10 data and program memory.

The AD-10 cross assembler generates object modules which are loadable by ADX. The assembler produces a comprehensive program listing, which includes the object code generated for each source statement, a symbol cross reference table and a detailed timing analysis of all multibus transactions which occur during program execution. The timing analysis and multiprocessor features of the assembler simplify the problems of relating the code associated with each processor. Conventional procedural reasoning is still valid in writing programs, even though this might not be expected because of the multiple processors.

User programming, at the assembly level, is simplified by an extensive library of application macrofiles. A macrofile is an AD-10 assembly language routine, in source file form, which can be included in a user application program by specifying input/output parameters or arguments. These routines, optimized for speed and accuracy, account for the great majority of code in most applications.

After assembly a program can be fully simulated in the host computer by the AD-10 simulator. This permits full debugging and program evaluation within the host. The simulator uses ADX for its control so the normal operating environment is preserved in the use of the simulator.

A high level simulation language for the AD-10 is now under development. This language, while having a few constraints not found in other continuous system simulation languages, will allow users to take full advantage of the AD-10 speed with minimum programming effort. The system will emphasize interactive involvement with the model and graphic display of solutions. It will be possible to optimize integration formulas, change scaling, change model parameters and initial conditions, document models and studies, and prepare hardcopy graphics. The AD-10 simulation language will also be compatible with a host simulation language so that the host computer can be used to generate test solutions and scaling information.

EXAMPLE

To illustrate the computational speed of the AD-10 multiprocessor described in this paper, the equations necessary to simulate an aircraft in six degrees of freedom were considered, including extensive aerodynamic functions.[3] One evaluation of all twelve state variables derivatives requires 47 adds, 108 multiplies, and 11 trigonometric functions in addition to computation of 2 two-variable, 8 three-variable, and 7 four-variable functions. Using the Macrofile Library routines for function generation, coordinate transformations and data transfers, as currently available in the standard AD-10 software, the time required for a complete pass through all the equations is less than 100 microseconds.

If an extrapolative type of numerical integration routine is used, then only one pass through the equations is required per numerical integration step. Since operations in the NIP (numerical integration processor) go on concurrently with those in the ARP (arithmetic processor) it is reasonable to assume that numerical integration of the 12 state variables will not add significantly to the frame time. Thus we will assume 100 microseconds per numerical integration step. Using third-order Adams-Bushforth (a popular integration routine for real-time digital simulation), roughly 50 integration frames per cycle of the highest problem frequency are needed for dynamic accuracy of the order of 0.1 percent. Thus the period of the highest problem frequency would be 50 (100) or 5000 microseconds, which corresponds to 200

hertz. Since the highest frequency associated with the rigid-body equations of a typical high performance aircraft is about 1 hertz (the so-called short-period pitching mode) the conclusion is that the AD-10 can solve the flight equations at 200 times real-time.

If the aircraft simulation also includes the flight control system, and perhaps elastic structural modes, then frequencies up to 25 hertz and higher may be present in the problem. Even allowing for the increased equation complexity in this case, it is clear that the AD-10 should be able to handle the real-time simulation comfortably.

CONCLUSIONS

Design of the AD-10 multiprocessor, optimized for continuous system simulation, has been described. The computer is characterized by a large high-speed interleaved memory, very fast arithmetic processor speeds, many high-speed input-output channels, and operation under a host general-purpose digital computer. The AD-10 multiprocessor appears capable of handling the most demanding of real-time aerospace vehicle simulations. Computing speed is approximately two-orders of magnitude faster than the speed of comparably priced general purpose minicomputers in simulating continuous systems. Six-degree-of-freedom aircraft flight equations can be solved at up to 200 times real-time. Such faster-than-real-time solutions could be very cost effective in parameter optimization problems and problems involving stochastic inputs, where large numbers of repetitive solutions are required.

REFERENCES

1. Gilbert, E. O., "A Special Digital Computer for High Speed Function Generation in Hybrid and Digital Simulation," *Proc. 8th AICA Congress*, Delft, August 23-28, 1976, North-Holland Publishing Co., Amsterdam, pp. 471-82.
2. Gilbert, E. O., and R. M. Howe, "An Expanded Role for Function Generation in Dynamic System Simulation," *Proc. 1977 Summer Computer Simulation Conference*, Chicago, July 18-20, 1977, pp. 305-308.
3. "Application of the AD-10 to Six-Degree-of-Freedom Flight Equations," Applications Report, Applied Dynamics International, Ann Arbor, Michigan, June 1977.

Plasma simulation on the UCLA CHI computer system

by JOHN M. DAWSON, ROBERT W. HUFF and CHENG-CHIN WU
University of California
Los Angeles, California

INTRODUCTION

Computer simulations have played an important role in fusion energy research. Although the basic physical principles of plasma physics are well understood, the coupled physical phenomena are so complex that analytical analyses cannot be easily carried out. Moreover, laboratory experiments are costly. With the enormous power of modern computers, one naturally resorts to computer simulations to gain insights into physical phenomena, as well as to guide the design and planning of experiments.

In order to cope with the rich and disparate phenomena occurring in a plasma, a wide variety of simulation models are required.[1] The models may generally be classified as: particle models, which treat the plasma at the microscopic level by following the motion of a large number of particles; fluid models, which treat plasma on a more macroscopic level; and also hybrid models which are a mixture of the first two models.

It is obvious that one cannot do plasma experiments in a computer with the 10^{12} to 10^{19} particles involved in physical problems of interest. Still, one would hope to have as many as 1 million particles on a mesh system of 64^3 for particle codes. For fluid codes, one may want a system of 64^3 to have better resolution, especially when local structure is present.

This shows that plasma simulations require a large computing capability and therefore have only been carried out on large computers, such as the IBM360/91 or CDC7600. Nevertheless, few attempts have been made for systems as large as mentioned above, for they need several megawords of storage and a considerable amount of computer time for a significant run.

Lately, with the development of computer technology, microprogrammable specialized minicomputers appear to provide the possibility of doing some of the jobs effectively and economically. Because of a successful pilot study[2] initiated by John M. Dawson and Burton D. Fried of the UCLA Plasma Physics Group and Glen J. Culler of Culler/Harrison, Inc. (CHI), a CHI plasma simulation computer system is being installed at UCLA. Our preliminary results indicate that the system offers a significant improvement in performance and, most importantly, it reduces cost by about two orders of magnitude from the IBM360/91.

The CHI plasma simulation system consists of six microprocessors: one array processor (AP-120B, manufactured by Floating Point Systems, Inc.),[3] one macro-processor (MP-32 made by CHI)[4] and four I/O processors (IOP made by CHI). The AP does most of the calculations, the IOP moves data around, and the MP does scheduling and control. Mass data are stored on disk. The parallel processing among and within these processors accounts for most of the performance advantage of the system. Further details of the system are given in the third section of this paper.

Three plasma simulation codes have been tested on the system: a 2-½ dimensional electrostatic particle simulation code,[2] and both 2-½ and 3 dimensional ideal MHD fluid codes.[5] The basic structure of these codes is described in the third section, together with comparative performance data. Conclusions are presented in last section.

THE UCLA CHI COMPUTER SYSTEM

General description of the system

The configuration of the UCLA CHI computer system is shown in Figure 1. It includes a macro-processor (MP-32), an array processor (AP-120B) and four input/output processors (IOP). Data Throughput and capacity of each processor is summarized in Table I. The AP does high speed floating point arithmetic operations. Further descriptions of the AP are given later.

The MP is used for control, I/O, bookkeeping calculations, and integer arithmetic operations, and serves as the host computer for the AP. In the present system it supports an interactive mathematical system (Math System),[7] provides a text editor, file facilities, the necessary assemblers and linkers.

The IOP's are fixed program micro-processors with access to both AP and MP main data memories and the UNIBUS. Their primary function is to provide high level control over access to the disk drives. Each IOP functions as a disk formatter supporting any of the four disk drives. This allows the configuration of individual drive capacities from 5 million to over 60 million 38 bit words. Transfer time per word is under 4 μs. In addition to transferring data between the AP main data memory and disk, the IOP can also transfer data

TABLE I.—Data Throughput and Capacity of the AP-120B, the MP-32A, the IOP and disks of the CHI System

Macro Processor (MP-32A)

(a) A 16-bit fixed point processor with a 1/6 μs machine cycle.

(b) Memories:

64 words scratch pad data memory
512 words fixed instruction memory
64 words writeable instruction memory
65,536 words of 1/3 μs instruction and data memory (CD).

(c) Integer multiplication of two 16-bit words in 1/3 μs.

(d) Functions:

(i) Controls high-speed local station displays.
Display rate: 2 μs/point
< 2 ms for longest line
> 4000 characters/second.

(ii) Schedules IOPs and Array Processor.

(iii) Controls data transfer to modems (1200-9600 baud) and other PDP-11 compatible I/O devices over Universal I/O bus.

Disk-IOPs

(a) Disk capacity: 4 Trident T80 drives - over 16 million 38-bit words each.

(b) IOPs are fixed program micro-processors.

(c) Data transfer rate: Data on all four disks can be simultaneously transferred to/from AP-MD or MP-CD, each at 250,000 words/second. Any IOP can be used for memory transfer between AP-MD and MP-CD at 1,000,000 words/second.

Array Processor (AP-120B)

(a) A 38-bit floating point arithmetic unit with maximum speed of 12 megaflops.

(b) Memories:

(i) Two 32-word data pad memories
2560 words 1/3 μs fixed table memory
65536 words 1/3 μs data memory (MD)
512 words instruction memory

(ii) Data pad access is immediate. Table memory may be referenced every cycle, with an access delay of two cycles. Data memory may be referenced every other cycle, with an access delay of three cycles.

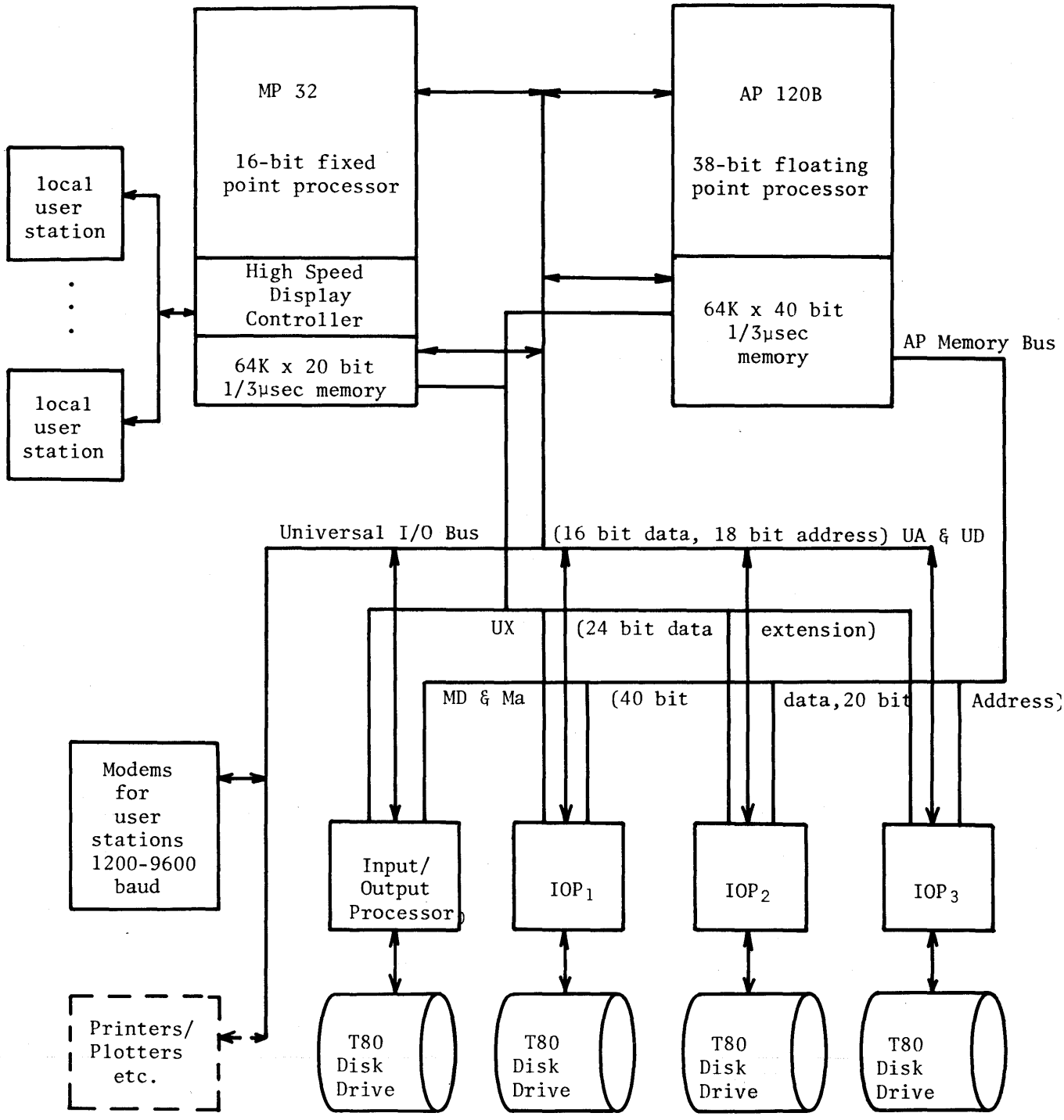


Figure 1—Block diagram of the UCLA CHI computer system

between the MP memory and disk or perform memory to memory transfers between or within either memory.

Each processor can process independently while the whole system is controlled by the MP (host processor). Each processor is scheduled to perform a specific process. It interrupts the host when it completes each process, then begins immediately, without having to wait for service from the MP, on its next process if one has been scheduled for it. This parallel logic computing capability is one of the main features of the system and makes possible the handling of large scale computing problems. An illustration is given in the third section.

AP-120B processor

Since all floating point calculations (therefore most of model calculations) are performed in the AP, it is of importance to explore the structure of the AP.[3] The architecture of the machine is shown in Figure 2; and its features are summarized in Table I.

Special features are noted as follows: (1) Programs, constants, and data each reside in separate, independent memories: program source memory, table memory and main data memory, respectively. Thus, it eliminates memory accessing conflicts. (2) Independent pipelining floating-point multiplier and adder units allow multiply and add operations to each be initiated every $^1/_6$ μs. (3) Two blocks of fast access floating-point accumulators, DPX and DPY with 32 locations each, are available for temporary storage of intermediate results from the multiplier, adder, or memory. (4) Address indexing and counting functions are performed by an inde-

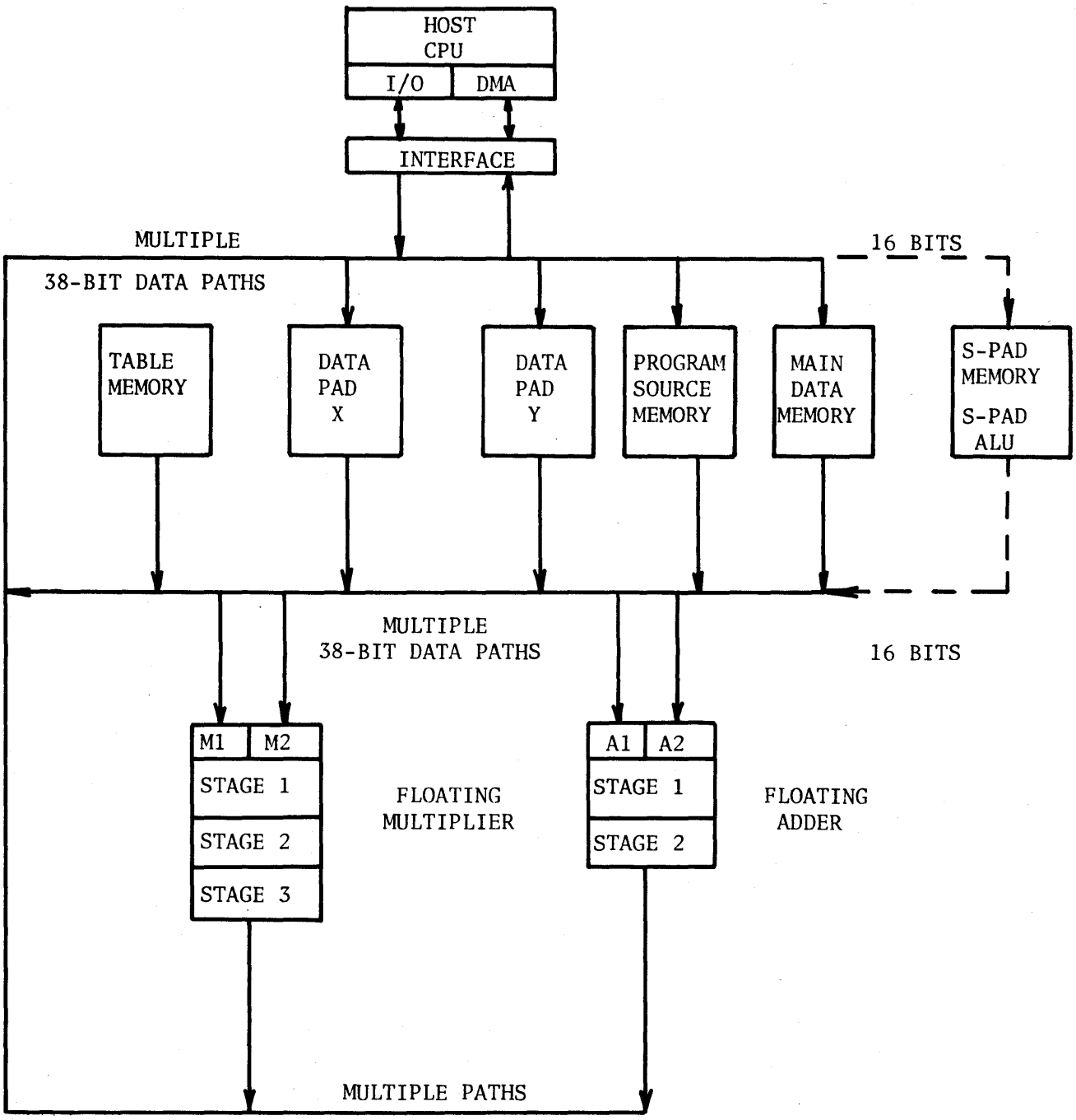


Figure 2—Block diagram of the AP-120B micro-processor (From Reference 3)

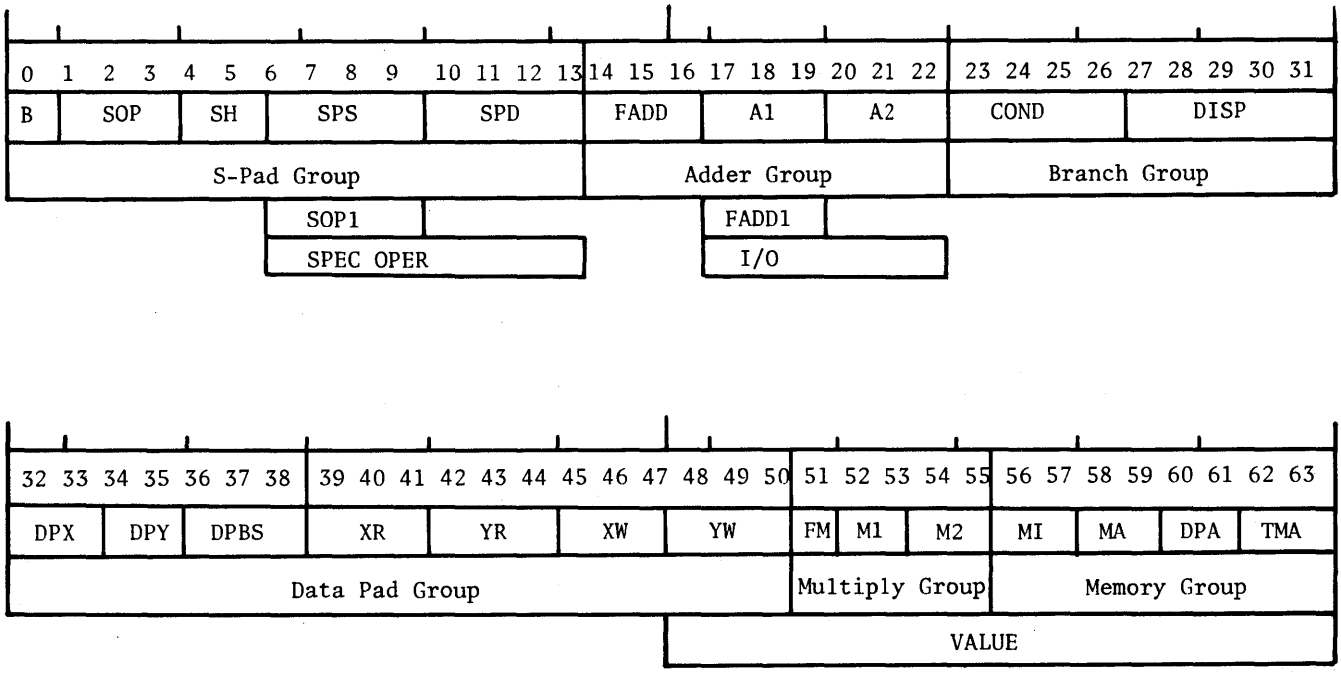


Figure 3—AP-120B Instruction Field Layout (From Reference 3)

pendent integer arithmetic unit (S-Pad and ALU). (5) Floating-point data is represented by 38-bit words which gives a precision of 8.1 decimal digits.

Each instruction word has 64 bits so that many different operations may be performed concurrently during each cycle. The instruction field layout is shown in Figure 3. It is separated into five general areas: program control, address control, arithmetic, memory and I/O. The highly parallel and pipeline structure of the AP-120B make it possible to achieve fast execution with lower speed logic and thus with lower cost. It requires $^1/_2$ μs for a single multiplication and $^1/_3$ μs for a single addition in the AP. But with adder and multiplier units working in parallel and with their pipeline structure, the AP allows multiply and add operations to each be initiated every $^1/_6$ μs. This gives a maximum speed of 12 megaflops comparable with a large general purpose computer. In the three test plasma simulation problems the speed achieved is about 4 megaflops for the fluid codes and 6 megaflops for the particle code.

In order to achieve this high speed capability, each instruction of a micro-coded program should use as many fields as possible. For this reason, our experience in coding plasma simulation codes suggests that calculations done in the manner of particle by particle, or grid-point by grid-point may be more efficient than trying to vectorize the code. Besides, vectorization usually requires several additional arrays for intermediate results. In contrast, vector operations are useful in applications such as digital signal processing, where the AP is called upon to perform vector operations at high speed. The Math System is designed for efficient support of these AP vector operations.

The basic arithmetic operations in the AP are addition and multiplication. The divide operation is carried out in software with speed of 3.83 μs. Hence, it is essential in the programming to use as few divide operations as possible. For instance, there is no divide operation in the $2^1/_2$D particle code; however, one such operation is required in the MHD codes. In the $2^1/_2$D MHD code, the divide operation uses 7 percent of the total processing time of 50 μs per grid point.

As previously mentioned, in the CHI system, the IOP's have a separate data bus into the AP main memory from that used by the AP processor. Thus the CHI system eliminates the bottleneck problem[8] which may exist in other host setups where all data must enter and leave the AP through the interface module and the host computer.

Programming languages

Programming can be carried out in four languages, which include two machine languages, the AP- and the MP- micro languages; and two high level languages, the MP-macro language[6] and the Math System language.[7] The AP micro-instructions have just been mentioned. The MP micro-instruction consists of 28 bits and provides for parallel operations. The MP macro-language is a set of macro-instructions which are pre-programmed in both the AP and the MP micro-languages and is used for system programming.

The Math System language, programmed in the MP micro- and macro-language is an interactive interpreter, and supports operations of mathematical functions applied to real or complex vector data as well as useful graphics. It can execute either interactively or from a program consisting of

```
(+) A, B → C        "C(i): = A(i) + B(i) for all i
(SIN) C             "C(i): = sin C(i) for all i
(DISPLAY)           "plot C(i) vs. i
```

Figure 4—A Math System program for the calculation, $C_i=\sin(A_i+B_i)$, with three vectors *A, B and C*

these mathematical operations. Since the Math System uses vector data which are entities usually occurring in physical problems, programming in this language is more concise and less error-prone than in the standard Fortran language.

A Math System program is shown in Figure 4 for the calculation of $C_i=\sin(A_i+B_i)$ with three vectors A, B and C. The operation $\oplus A,B\rightarrow C$ calculates $C_i=A_i+B_i$ for all i, so does the operation SIN C for $C_i=\sin(C_i)$. The program therefore saves an explicit loop that is required in a Fortran program. The statement DISPLAY will plot C_i against i in terms of its natural units.

In plasma simulation codes, we anticipate using only the Math System language and the AP micro-language. AP micro-programming is done only where the development of maximally efficient routines is warranted, as the particle push routines for particle simulations. The remaining routines, such as initialization, control, and force-calculation routines in the particle codes, which require only a small fraction of total processing time, are easily coded in the Math System language.

The operating system supports both batch and on-line usage. Thus, it allows old production or test runs of a code simultaneous with code development, editing or on-line Math System usage.

PLASMA SIMULATION MODELS AND THEIR TEST RESULTS ON THE CHI SYSTEM

A particle code and a fluid code, which together represent a wide spectrum of plasma simulation models, have been adapted to the CHI computer system. The particle code is the standard 2½ dimensional electrostatic model with a constant magnetic field plus mirroring.[2] The fluid code is an ideal magnetohydrodynamics (MHD) code,[5] which is being implemented in both 2½ and 3-dimensional versions. These are very basic models that include the most important general features of plasma simulations.

In this section, we present basic structures of these codes, the utilization of the resources of the CHI system and their test results. Details of the models and their numerical method aspects will not be discussed here.

A 2½ dimensional electrostatic model with a constant magnetic field

In a particle model, the trajectories of a large number of charged particles (electrons and ions) under the influence of self-consistent fields are followed in time. The particular model considered consists of a large number of charged rods parallel to the z-axis which are allowed to have velocity components in the x, y, and z directions as shown in Figure 5. It allows spatial variation in the x and y directions. The system is periodic in both x and y directions such that particles that leave one side of the system are re-entered at the opposite side. A uniform static magnetic field is permitted to point in an arbitrary direction. The charged rods move under the influence of their self-consistent electric fields and the given magnetic field. In addition, the rods may be reflected at the y-boundaries if a mirroring condition is satisfied by the velocity components parallel and perpendicular to the magnetic field.

The motion of the system is generated in the computer by a leap-frog time difference scheme, using a standard finite-size particle model, in which the charge of a particle is distributed over a finite region about the center of mass, and also using a dipole expansion technique. The finite-size particle model effectively reduces undesirable collisions and also numerical noise from the discrete nature of the spatial grid. However, in the following discussions, only the basics of the model are presented.

Mathematically, the simulation model solves the following equations self-consistently:

$$\vec{\nabla}\cdot\vec{E}(\vec{r})=\rho(\vec{r}), \tag{1}$$

$$\frac{d\vec{v}_i}{dt}=\frac{q_i}{m_i}\left(\vec{E}(\vec{r}_i)+\vec{v}_i\times\vec{B}(\vec{r}_i)\right), \tag{2}$$

$$\frac{d\vec{r}_i}{dt}=\vec{v}_i. \tag{3}$$

$\vec{E}$ and $\vec{B}$ are the electric and magnetic fields, respectively; $\vec{B}$ is assumed to be a constant vector. ρ is the charge density distribution and $\vec{r}_i$, $\vec{v}_i$ stand for the position and velocity of the ith particle with charge q_i and mass m_i.

In the 2½D model, a spatial grid mesh in the x-y plane

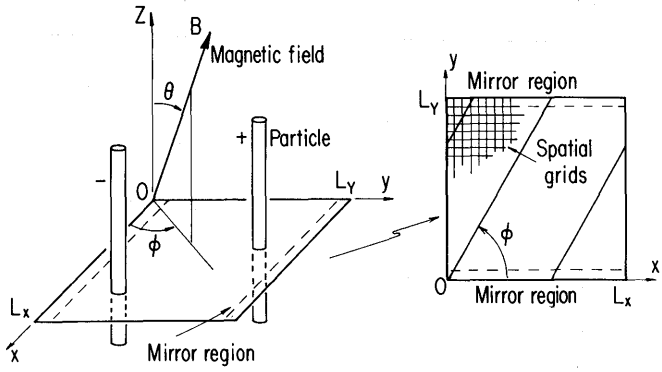
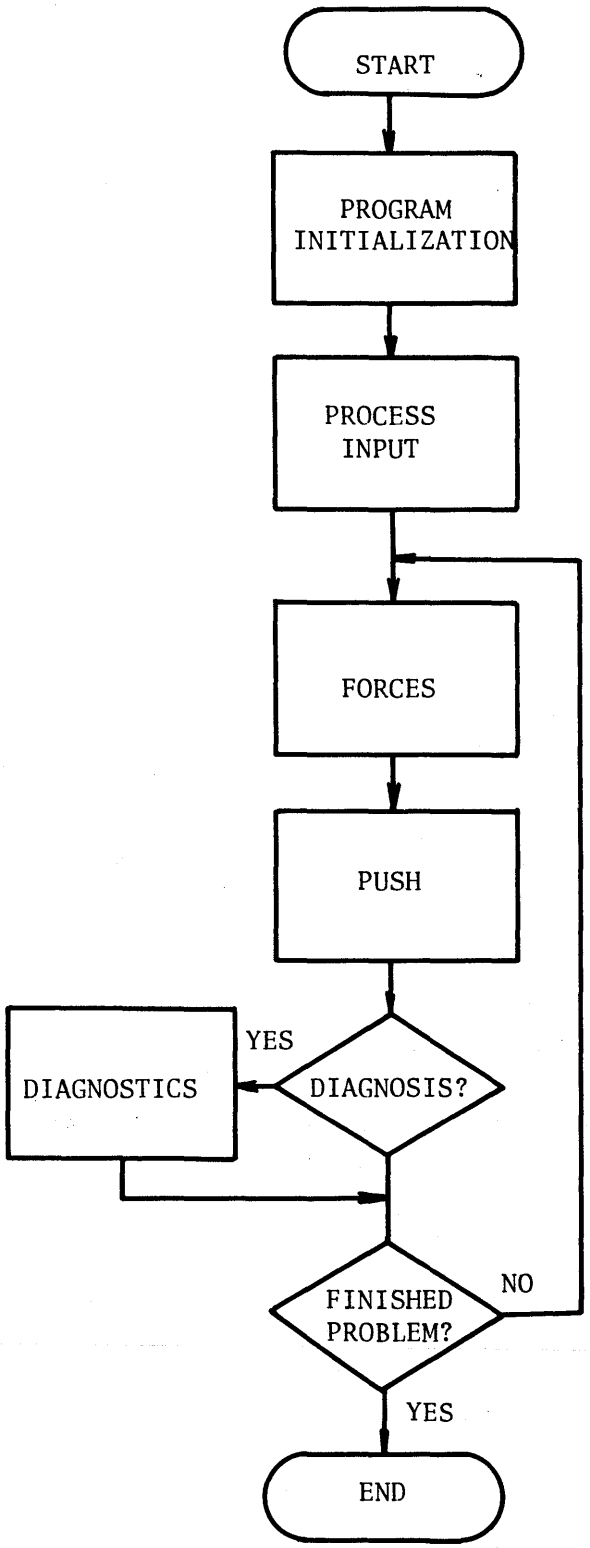


Figure 5—2-½ D Plasma particle model with mirror



Initialize: (1) Random Numbers
(2) Disk Variables
(3) Control Parameters

(1) Read input and generate constants.
(2) Compute initial position, velocity for every particle.
(3) Compute initial charge distribution.

Compute forces on grid points from charge distribution using Fast Fourier Transform.

IOP, AP Work in parallel to:
(1) Update position, velocity of each particle.
(2) Compute new charge distribution.
(3) Compute energy for error monitoring.

Computes one or more of:
(A) Temperature
(B) Diffusion
(C) Flux-tube charge density.

Figure 6—Program flow of the particle model

is introduced. All macroscopic quantities, such as $\vec{E}$ and ρ are calculated on the grid points. The basic flow of the program is shown in Figure 6. It starts with setting up initial configuration of the system by assigning $\vec{v}_i$ and $\vec{r}_i$ and q_i for each particle and specifying the $\vec{B}$ field.

Inside the loop, calculations are performed in two stages: (a) Force calculation: $\rho(\vec{r})$ is used to solve Eq. (1) for the electric field $\vec{E}$. Eq. (1) is solved efficiently by fast Fourier transforms. (This technique is especially useful when finite-size particle model is used.) (b) Particle push: After $\vec{E}$ is obtained, the particle positions and velocities are advanced in time by means of Eqs. (2) and (3), using the leap-frog method with a dipole expansion of the force. As each particle position is updated, its contribution to $\rho(\vec{r})$, including dipole terms, is calculated. We are now ready to start the next time step.

In the actual code with the finite size particle model and dipole expansion technique, the data required are 6 words per particle, i.e., $\vec{r}$ and $\vec{v}$ with three components each, 9 words per grid points which include E_x, E_y, ρ, D_x, D_y, E_{xx}, E_{xy}, E_{yx}, and E_{yy}. (The last six variables are dipole terms.) Since particles are grouped according to species, it is not necessary to specify charge and mass for each particle individually.

For a 10^6 particle and 64^2 grid-point system, more than 90 percent of the computer time is used in the particle PUSH routine. Therefore the PUSH routine is coded in the AP micro-language with maximal optimization, whereas other routines are efficiently programmed in the Math System. The PUSH routine takes 13 μs per particle which is four times the speed of the corresponding IBM360/91 assembly-language code. The AP PUSH routine is not a direct translation of the corresponding Fortran code, but a new design for an efficient usage of the AP. There are 33 multiplications and 45 additions in the routine, and therefore an effective speed of 6 megaflops is achieved.

The data storage requirement as well as processing timing are given in Table II. To handle the large quantity of data the technique of parallel processing among the AP, the MP and the IOP's is used. To further explain this parallelism, a section of the Math System program is shown in Figure 7. The section is essentially the control program for pushing particles. The actual code, however, is a simple three-statement loop program for this two-disk system.

The instruction, OP PUSH BUFFI causes the AP to initiate or schedule an AP micro-program named PUSH for advancing particles whose descriptors are in the array BUFFI of AP main memory. The execution will begin as soon as the AP has completed all its previously scheduled processes and data BUFF1 is free from operations associated with other processors. If the AP is ready but data block BUFF1 is still being loaded with data by the IOP, the AP will wait until the latter action is completed. The instruction, MOVE INDISK→BUFF1, will instruct an IOP to move data from disk that is used for input to the location BUFF1 in AP main memory. An IOP will perform the operation once it comes to the instruction and neither block of data is being used by other processors. Similar operation is defined for the instruction MOVE BUFF1→OUTDISK.

```
Math System Program                          Processor

. . .

MOVE   INDISK → BUFF3                        IOP1

MOVE   BUFF1 → OUTDISK                       IOP2

OP     PUSH    BUFF2                         AP

MOVE   INDISK → BUFF1                        IOP1

MOVE   BUFF2 → OUTDISK                       IOP2

OP     PUSH    BUFF3                         AP

MOVE   INDISK → BUFF2                        IOP1

MOVE   BUFF3 → OUTDISK                       IOP2

OP     PUSH    BUFF1                         AP

. . .
```

Figure 7—A section of a Math System program for controlling the AP and the IOP's in particle push

Therefore, in executing the program shown in Figure 7, two IOP's and the AP are operating simultaneously: One IOP loads into one block of AP main memory a block of old particle data from one disk, the AP executes PUSH routine in advancing particle data stored in the second block of AP memory, while the other IOP stores a third block of updated particle data from AP memory into the second disk. By choosing adequate blocksize and using enough number of IOP's, it is possible to eliminate restrictions on over-all processing time imposed by I/O.

To handle 10^6 particles, four disks are used with two disks for input, and two disks for output. The two pair of disks are interchanged with respect to I/O at each time step. The size of block is chosen to be 4K words (a track or 680 particle descriptors). It takes 17.7 ms for the AP to update two blocks of data and 20 ms for each IOP to transfer one track of data between disk and AP memory. (The transfer rate of 20 ms per track includes a 20 percent allowance for the loss of one rotation at the cylinder boundary). The over-all processing time is then slightly limited by I/O to 14.7 μs per particle.

Magnetohydrodynamic codes

The set of magnetohydrodynamic equations includes:

$$\frac{\partial \rho}{\partial t} = -\vec{\nabla}\cdot(\rho\vec{v}) + \epsilon\nabla^2\rho, \tag{4}$$

$$\frac{\partial}{\partial t}(\rho v_i) = -\frac{\partial}{\partial x_j}\left(\rho v_i v_j + \left(\rho T + \frac{B^2}{2}\right)\delta_{ij} - B_i B_j\right) + \nu\nabla^2\rho v_i - \lambda\rho v_i - \rho\vec{g}, \tag{5}$$

$$\frac{\partial \vec{B}}{\partial t} = \vec{\nabla}\times(\vec{v}\times\vec{B}) + \eta\nabla^2\vec{B}, \tag{6}$$

TABLE II.—Resource Utilization and Processing Timing for a 10^6 Particle and 64^2 Grid Point Particle Simulation Code on the CHI system

Data:

6 Words/particle x, y, z, v_x, v_y, v_z

9 Words/gridpoint: E_x, E_y, E_{xx}, E_{xy}, E_{yx}, E_{yy}, C, D_x, D_y

The particle descriptors are on disk, 680 particles/track, 1472 tracks, or 736 tracks on each of two disks for input, and the same for output, with the two pairs of disks being interchanged with respect to I/O at each timestep.

AP-Memory Allocation:

Electric Field arrays (6):	24k
Charge and dipole arrays (3):	12K
Particle buffers (5):	20k
Total	56k

Data Transfer Timing:

60 disk rotations/second or 16.7 ms/track, for a transfer rate of 24.5 μs/particle for each of two disks, or an overall rate of 12.2 μs/particle on a track-by-track basis. Loss of one rotation at the cylinder boundary (5 tracks/cylinder) increases this by 20%, giving 15 μs/particle on a cylinder-by-cylinder basis.

Processing Time:

Dominated by the particle-push AP routine for this system size.
Present AP code estimated at 13 μs/particle.
Thus the AP processing is slower than the I/O on a track-by-track basis, but is faster than the I/O on an overall basis.

Overall Timing:

15 μs/particle overall, or 15 seconds/timestep.
16 hours for a run of 4000 timesteps (to $\omega_p t = 1000$ @ $\Delta t = 0.25/\omega_p$)

TABLE III.—Resource Utilization and Processing Timing for a 2-½ D MHD Code

Data:

7 variables: $V_x, V_y, V_z, B_x, B_y, B_z, \rho$.

Each variable occupies an array of dimension $(N + 3)*(N_y + 2)$, where N_x and N_y are the numbers of grids in x- and y-direction, respectively.

The system is periodic in y and bounded in x. Temperature is assumed to be constant.

AP-Memory Allocation:

$7*(N_x + 3)*(N_y + 2) + 30$ words.

With 45K memory, the system can be as large as 80 x 80.

AP-Program Memory:

445 program words.

Processing Time:

55 μsec/grid for a 20 x 20 system.

50 μsec/grid for a 40 x 40 system.

There are $1/2*(N_x + 1)*N_y$ grid calculations per time step in the leap-frog scheme.

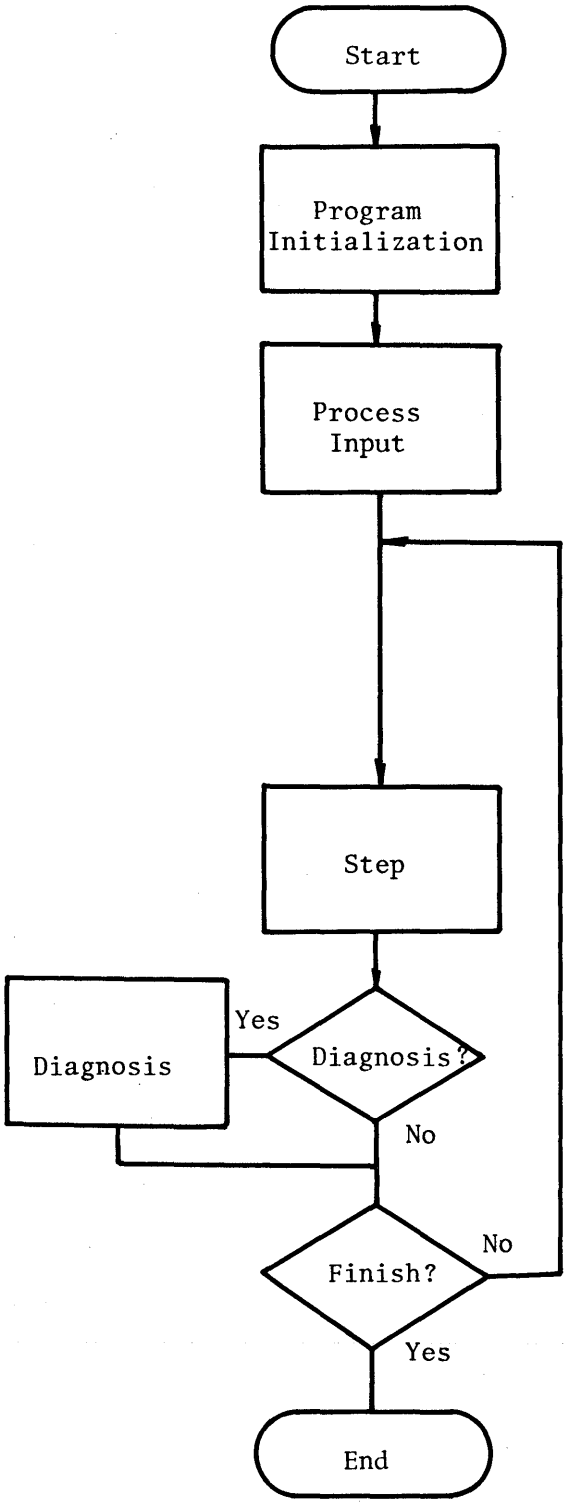
$$\frac{\partial T}{\partial t} = -\vec{\nabla}\cdot(T\vec{v}) + (2-\gamma)T\vec{\nabla}\cdot\vec{v} + \frac{\eta(\gamma-1)}{\rho}(\vec{\nabla}\times\vec{B})^2 + (\gamma-1)\nu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \delta_{ij}\vec{\nabla}\cdot\vec{v}\right)\frac{\partial v_i}{\partial x_j} + (\gamma-1)\kappa\nabla^2 T. \quad (7)$$

Here, the mass density is denoted by ρ, the velocity by $\vec{v}$, the temperature by T, the magnetic field by $\vec{B}$ and the gravitational acceleration by $\vec{g}$. γ is the ratio of specific heats. The transport coefficients ϵ, ν, λ, η and κ are assumed to be constants.

The computer code solves the equations as an initial-value problem, by integrating them in time using an Eulerian grid and an explicit, conservative, leap-frog finite-difference scheme. The details of the numerical procedures can be found elsewhere.[1,5]

All variables are macroscopic and are defined at grid points. In a three-dimensional code, a three dimensional grid is introduced. For the case of 2½ dimensional code, a two dimensional grid is used, with $\vec{v}$ and $\vec{B}$ still having three components.

The basic flow of the program as shown in Figure 8 is a loop over timestep, similar to that of particle model, except here the routine STEP performs one timestep integration of the MHD equations for all grid points. The routine STEP is coded in the AP micro-language, whereas other routines including initialization and control are in the Math System.



Initialize:
(1) Control parameters
(2) Disk variables

(1) Compute initial plasma configuration, set up initial values for ρ, $\vec{v}$, $\vec{B}$ and T
or

(2) If restart, read input from disks

AP and IOPs (if any) work parallel to integrate MHD equations from time t to t + Δt over all grid points.

Compute various energies, output ρ, $\vec{v}$, $\vec{B}$ and T distributions.

Figure 8—Program flow of the MHD fluid codes

TABLE IV.—Comparison between the CHI System and the IBM360/91 with Respect to the Three Test Plasma Simulation Codes

	IBM360/91	CHI SYSTEM
PARTICLE CODE		
Time/Particle Push	Assembly Code	
CPU	55 μs	13 μs
I/O	- -	0.6 × 24 μs
Overall	55 μs	14 μs
10^6 Particles, 4000 Timesteps		
Time	61 hr	16 hr
CPU Cost	32 k$	- -
I/O Cost	18 k$	- -
Total Cost	50 k$	0.3 k$
2-1/2D MHD CODE	Fortran Code (H-Compiler)	
Time/Gridpoint	130 μs	50 μs
3D MHD CODE	Fortran (H-Compiler)	
Time/Gridpoint	230 μs	90 μs
64^3 System, 4000 Timesteps		
Time	33 hr	13 hr
CPU Cost	17 k$	- -
I/O Cost	10 k$	- -
Total Cost	27 k$	0.24 k$

The routine STEP is coded in such a way that calculations are done grid-point by grid-point, which seems to be the most efficient method in utilizing parallel operations in the AP. The processing time for the $2^1/_2$D code is 50 μs per grid as compared with 130 μs for a corresponding Fortran code on IBM360/91. For the 3D code, it is 90 μs on the CHI computer, while a Fortran code requires 230 μs on the IBM360/91. The number of floating point operations is 101 multiplications, 92 additions and 1 division for the $2^1/_2$ code and 184 multiplications, 174 additions and 1 division for the 3D code. Both codes reach an effective speed of about 4 megaflops.

The storage requirement, processing timing and other statistics of the $2^1/_2$D code is given in Table III. This code is an isothermal one, in which temperature is a constant, so there are seven equations instead of eight. The AP main memory is big enough to store data for a system as large as 80^2.

In case of the 3D code, disks must be used; for instance, it requires two megawords of data in a 64^3 system. In each grid point calculation, eight variables defined on the grid point and its six nearest neighbor points, or a total of 56 numbers are used. However, some of these data can be saved for calculations at the next grid point in the column or adjacent grid point in the next column, so the need of data transfer is reduced. The technique of parallel processing between the IOP's and the AP is used.

Comparison with results on the IBM360/91

The comparison of these plasma simulation codes on the CHI system and the corresponding Fortran (or Assembly) codes on the IBM360/91 are given in Table IV. For executing a one-million-particle simulation or a 64^3 3D MHD code on the IBM360/91, we assume disks are used for storage. The I/O charges are based on this assumption.

The speed of the particle code is four times faster than the IBM assembly-language code. For the MHD codes, the speed advantage is not as good. Nevertheless, all codes attain an effective speed of 4 to 6 megaflops. Most importantly, the CHI system shows a tremendous savings in total cost, with better than a two-order-of-magnitude decrease in cost from the IBM360/91. The figures for the IBM360/91 are actual costs if the programs were run at UCLA. For the CHI computer system, estimates are based on the assumption that computer time charge is directly proportional to the price of the computer system.

In addition, we should point out that the CHI system achieves better accuracy than the IBM360/91 in single precision. One example is that the kinetic energy calculated in the $2^1/_2$D MHD code on the CHI machine is two digits more accurate than on the IBM360/91 in single precision.

CONCLUSION

We have presented a general description of the UCLA CHI computer system and have emphasized its special features, namely, two levels of parallelism in the system. Parallel processing among the AP, the MP, and the IOP's allows execution of large-scale problems, and a high degree of internal parallelism within each processor allows fast operations at very moderate cost. In addition the CHI system provides a time sharing operating system and supports a high level language, the Math System, which is particularly useful in solving physics related problems.

The test results for three plasma simulation models clearly show the capabilities of the system. It performs high speed execution of large problems at relatively low cost. Since the plasma simulation models are similar in structure to many simulation models in other research fields, the CHI computer system is expected to have a wide range of applications.[8]

ACKNOWLEDGMENTS

The authors gratefully acknowledge discussions with our colleagues at UCLA and CHI: G. Ball, G. J. Culler, M. Curtis, B. Fried, P. Kokelaar, M. McCammon, and P. L. Pritchett.

This work was supported by ERDA, Contract No. EY-76-C-03-0010 PA26, Task III.

REFERENCES

1. For a general discussion of plasma simulation models, refer to "Methods in Computational Physics," edited by B. Adler, S. Fernbach, and M. Rotenberg, Vol. 16, Academic, New York, 1976.
2. Kamimura, T., J. M. Dawson, B. Rosen, G. J. Culler, R. D. Levee and G. Ball, Plasma Simulation on the CHI Micro-processor System," UCLA Report, p. 248, 1975.
3. AP-120B Processor Handbook, Floating Point Systems, Inc., Portland, Oregon, Form #7259, 1976.
4. MP-32A Micro-Programming Manual, Culler-Harrison, Inc., Goleta, CA., Publication 29107, 1973.
5. Pritchett, P. L., K. V. Roberts, and J. M. Dawson, *Bull. Am. Phys. Soc.*, 21, p. 1045, 1976. K. V. Roberts and D. E. Potter, in "Methods in Computational Physics," edited by B. Adler, S. Fernbach, and M. Rotenberg, Academic, New York, 1970, Vol. 9, p. 339.
6. MP-32A Macro-Programming Manual, Culler-Harrison, Inc., Goleta, CA., Publication 29106, 1973.
7. The Math System is an extensive generalization of earlier systems developed at UCLA, UCSB and TRW by G. Culler and B. Fried. User's Manual, Culler-Harrison, Inc., Goleta, CA., 1977.
8. Karplus, W. J., "Peripheral Processors for High-Speed Simulation," *Simulation* 29, p. 143, 1977.

Multiple microcomputer systems for solving certain fluid flow problems

by JOHN STEINHOFF
Grumman Aerospace Corporation
Bethpage, New York

INTRODUCTION

In the last several years, minicomputer based systems have been considered or used for an increasing number of scientific computing tasks. For example, Kottler and McGill[1] used a minicomputer system to solve a series of problems, including a partial differential equation for pollutant dispersal. Also, in 1973, Steven Orszag[2] compared a minicomputer system to large general purpose machines for solving some time dependent hydrodynamics codes. The conclusions were that minicomputers could be more cost effective than the large machines. In particular, Orszag concluded that the minicomputer was more cost effective than a CDC 7600 but less than a CDC STAR for moderate resolution hydrodynamics codes. For these problems, the data bases are large and the system is essentially a disk based one, where data flows from a large data base on the disk to the computer. When these studies were done, disk speed was not a restriction and the limiting factor was the computational speed of the minicomputer.

In the last several years, improvements in LSI technology have made it possible to assemble reliable computational devices with a much lower cost per computation than minicomputers. These devices, ranging from moderately priced (~$100,000) fast systems with tightly coupled multiple functional units to very inexpensive simple units can be very cost effective when used with a minicomputer as host. They have had a large impact on the signal processing field,[3] in some cases proving to be faster than large machines such as the CDC 7600. They can also be effective for real-time simulation, as described in Reference 4.

In this paper we will describe a study that was done on the effectiveness of a minicomputer based system with moving head disk and peripheral processor for a class of fluid flow problems.

It is easy to see that a moderately priced system can be assembled that would have a (disk) data capacity and contiguous block throughput and a (peripheral) computational speed of a large multi-million dollar machine. The extent to which this potential can be realized for our problem depends on, among other things;

- Whether data can be transferred between disk and a relatively small random access memory at close to contiguous-block rate during the computation.
- Whether the elements of the peripheral processor can be kept operating efficiently (in parallel).
- Whether the system can be programmed efficiently.

We will describe a disk data allocation scheme that avoids excessive disk accesses and achieves a high data throughput rate, and a scheme to use a set of peripheral processors efficiently, for the problem considered. We will also describe progress that we have made in assembling and operating a prototype system. At this time we have not obtained enough programming experience to comment on the ease of use of the system. The types of problems that we are looking at will be run for long periods of time, justifying the extensive use of assembly language programs. Once the possibility of effectively solving these problems has been established, it will be worthwhile to develop software routines which can be used in higher level language programs.

Finally, we will describe a scheme to assemble a very fast system consisting of a set of modules arranged in a simple configuration. This system will have the capacity and speed to economically solve some very large problems.

DYNAMICAL EQUATIONS

The basic problem involves the solution of turbulent flow of a viscous, incompressible fluid, using the Large Eddy Simulation approach. In this approach, the Navier Stokes equations which describe the detailed dynamics are first spatially filtered resulting in a damping of the high spatial frequency modes of the turbulent flow. After removal of these modes, which cannot be resolved on feasible computers, only the resolvable "large eddies" are treated explicitly. A sub-grid model is used to compute the stresses caused by the removed "small eddies." The resulting equations, when discretized, are tractable on current very large computers. The approach leads to good predictions for some quantities, such as the filtered energy spectrum in isotropic turbulent decay.

Our initial approach will be similar to that of the Stanford Group.[5,6] Most of this work has been done on a CDC 7600 and we should have a good basis for comparison.

The equations that we are solving are

$$\frac{\partial u_i}{\partial t} = v_i - \frac{\partial P}{\partial x_i} \tag{1}$$

$$\frac{\partial^2}{\partial x_i^2} P = \frac{\partial}{\partial x_i} v_i \tag{2}$$

$$v_i = -\left(1 - C_\Delta \frac{\partial^2}{\partial x_k^2}\right) u_j \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) - \frac{\partial}{\partial x_j} \tau_{ij} + \nu \frac{\partial^2}{\partial x_j^2} u_i \tag{3}$$

where u_i are the filtered velocities, P is the pressure, ν is the molecular viscosity, τ_{ij} the model term for the high frequency modes and C_Δ is a constant depending on the filtering function. The most popular model for τ_{ij} is based on the Smagorinsky model:

$$\tau_{ij} = -2\nu_T s_{ij}$$

$$s_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

$$\nu_T = c(2 s_{ij} s_{ij})^{1/2}.$$

In the above equations, repeated indices imply summation.

The basic approach is to use (1) and (3) to compute u_i at a new time step, with P computed using (2). This method enforces incompressibility by predicting zero divergence for u_i at a new time step, if u_i has zero divergence at the current time step. The Adams-Bashforth method, which is second-order accurate in Δt, is used to integrate u_i:

$$u_i^{(n+1)} = u_i^{(n)} + \Delta t \left[\frac{3}{2}\frac{\partial u_i^{(n)}}{\partial t} - \frac{1}{2}\frac{\partial u_i^{(n-1)}}{\partial t}\right]. \tag{4}$$

Various finite difference methods have been used to approximate the spatial derivatives. Our example will involve second order equations for the filter related terms (containing C_Δ) and fourth order terms for all other derivatives. Other difference formulae, involving a fixed number of neighboring points, can be computed on our system with schemes similar to the one we will describe. In one dimension, the approximations to the first derivative are:

$$\delta a(x) = \frac{1}{2\Delta x}[a(x+\Delta x) - a(x-\Delta x)]$$

$$\delta a(x) = \frac{1}{12\Delta x}[-a(x+2\Delta x) + 8a(x+\Delta x) - 8a(x-\Delta x) + a(x-2\Delta x)]$$

Our initial problems will involve homogeneous flows. The periodic boundary conditions used in these flows are simple to handle and represent a first study for our system. Other boundary conditions, such as no-slip ones, result in additional problems from the numerical analysis[7] as well as programming point of view. Especially when higher order methods are used, different operations must be done at grid points near boundaries. Unlike array processors, our system will be able to do a number of these different tasks in parallel, but the programming will be more difficult. For the periodic boundary conditions used, the Poisson equation for the pressure can be solved directly using three-dimensional Fourier transforms.

COMPUTING SYSTEM

The basic computing system considered for the study consists of a moving head disk, a minicomputer and a set of bipolar microcomputers connected to the minicomputer by a time-multiplexed bus (see Figure 1). A prototype system is currently being constructed in the Research Department at Grumman.

The capacities of available minicomputer disks run up to 300 megabytes (Trident T-300) at a cost of about $20,000, with (contiguous block) transfer rates up to about 500,000 bytes per second. The access times are about 10 to 70 milliseconds depending on how many tracks the head must pass over. The particular disk used in our system has a 10 megabyte capacity and 160,000 byte/second transfer rate. For the problem considered, the disk characteristics are very important.

The host minicomputer used in the prototype system is a Data General NOVA 800, with a random access memory of 32,000 16 bit words and an 800 ns. basic cycle time. It is not clear whether a higher performance minicomputer would result in higher throughput for our problem. Although the NOVA 800 cost more when purchased, minicomputers with similar performance now cost less than $10,000.

The peripheral processor considered is of the type being constructed in the Research Department. It consists of a set of independent 16 bit machines, each with a cycle time of 180 ns., separate high speed cache data memory (1K word) and microprogram memory (2K word) together with a larger 16-32K MOS data memory. For our problem, no direct intermodule communication is necessary and a single time-multiplexed data bus allows each microcomputer, when enabled, to communicate to the host via direct memory access (DMA). The number of microprocessor modules that can be

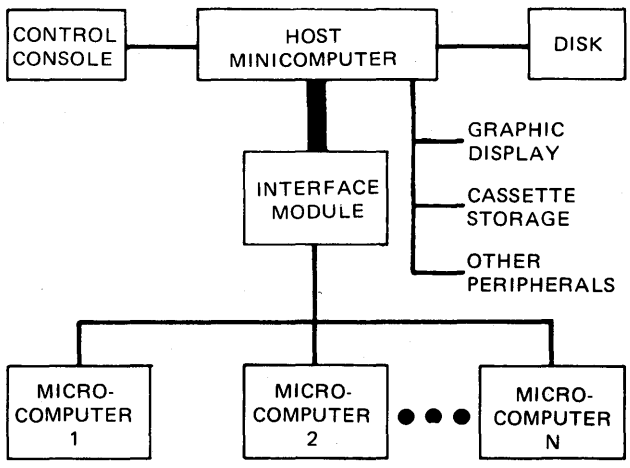


Figure 1—Computing system organization

used effectively depends mainly on the disk transfer rate for available disks. The cost of each module is about $4,000.

IMPLEMENTATION

Data flow

The grid that we are using contains $(64)^3$ nodes (the largest size run on the 7600) and the resulting data base for the problem is at least several million words, depending on how it is decomposed. Hence, for a modest system such as ours a moving head disk must be used for data storage. For available disks, the access time is three orders of magnitude longer than the time required to transfer a (32 bit) word. Also, this transfer time is an order of magnitude longer than the read time for the random access memories used in the system. Thus it is essential to use a data allocation scheme that minimizes disk accesses, and an algorithm that minimizes the total number of variables transferred to and from disk each time step.

For our problem, we can reduce the disk variables to two three-vector arrays and a scalar array:

$$\tilde{u}_i^{(n)},\ v_i^{(n)} \text{ and } \tilde{P}^{(n)},$$

where

$$\tilde{u}_i^{(n)} = u_i^{(n)} + \tfrac{3}{2}\Delta t v_i^{(n)} - \tfrac{1}{2}\Delta t v_i^{(n-1)} + \tfrac{1}{2}\Delta t \frac{\partial}{\partial x_i} P^{(n-1)}$$

and $v_i^{(n)}$ and $P^{(n)}$ are defined in (3) and (2). Then,

$$u_i^{(n+1)} = \tilde{u}_i^{(n)} - \tfrac{3}{2}\Delta t \frac{\partial}{\partial x_i} P^{(n)}$$

In the above, $\tilde{P}^{(n)}$ is the one-dimensional Fourier transform of $P^{(n)}$ (the reason for storing $\tilde{P}^{(n)}$ and not $P^{(n)}$ will be explained below), and n refers to the time step.

The computations required at each time step can be divided into two classes: a semi-local class where finite difference approximations to the partial derivatives are computed and the velocities are updated; and a global class where Poisson's equation is solved by a direct method. The data flow requirements for these two classes are quite different. Data allocation and computational schemes have been formulated that require a relatively small number of disk accesses and a near-minimum number of variables to be read from and written to disk for these classes of computations.

First, the data allocation scheme will be described. The $\tilde{u}_i$, v_i and $\tilde{P}$ arrays are partitioned into "records" that span 4 units (out of 64) in the x_1 direction, 8 units in the x_3 direction and all 64 units in the x_2 direction (see Figure 2). These records are arranged on disk in 16 segments (8 records each) that correspond to different sets of x_1 values and all 64 x_3 values, together with extra "scratchpad" space (for two records). Although the segments are arranged sequentially on the disk, they are depicted in Figure 2 next to each other. In the figure, the index represents the position of the record in the x_1-x_3 plane. The positions are shown for a particular stage in the sweep through the $\tilde{u}_i$ array, to be described below.

There are three computational phases at each time step. In the first phase, $\tilde{u}_i^{(n)}$, $v_i^{(n)}$, $\tilde{P}^{(n)}$ are read from disk and $\tilde{u}_i^{(n+1)}$, $v_i^{(n+1)}$ written back. This phase requires 8 sweeps in the x_1 direction (for different x_3 values), each sweep taking 16 steps (see Figure 3). At each step a set of $\tilde{P}_i^{(n)}$ values are read from disk and $P^{(n)}$ computed by Fourier transform in the x_2 direction. Then $\delta_i P^{(n)}$ is computed, $\tilde{u}_i^{(n)}$ read and $u_i^{(n+1)}$ computed. Then τ_{ij} and $v_i^{(n+1)}$ are computed. Finally, $v_i^{(n)}$ is read from disk and $\tilde{u}_i^{(n+1)}$ computed and written back to disk together with $v_i^{(n+1)}$. It is easy to see that, since each record contains all 64 values of x_2, once these are in random access memory any partial derivatives as well as Fourier transforms can be computed in this direction. Also, x_1 derivatives can be computed efficiently by storing some values from the previous steps since the sweep is in the x_1 direction. Only for the x_3 derivatives do we need to read some variables more than once: As pictured in Figure 3 for each sweep, a band must be read that is wider than the width of the sweep ($8x_3$ units). Also, we cannot write over variables during a sweep if the old ones are needed again for the next sweep. This is illustrated in Figure 2, where the positions of the updated $\tilde{u}_i^{(n+1)}$ records and the old $\tilde{u}_i^{(n)}$ records are shown with our scheme. Only one disk access is needed to read the $\tilde{u}_i^{(n)}$ variables required for the next step (including overlays) and write the updated ones. Also, at each step one access is required to read the $\tilde{P}^{(n)}$ variables (including overlap) and one to read and write the $v_i^{(n)}$ variables. (Steps near the edges of the grid require extra accesses because of the periodic boundary conditions).

The second phase required at each time step involves computing $\rho^{(n)} \equiv \delta_i v_i^{(n)}$, the source function for Poisson's equation (2). The procedure used in the first phase is used

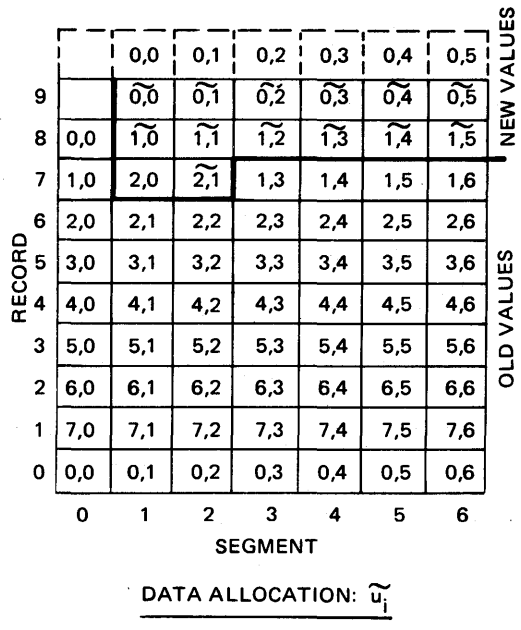


Figure 2

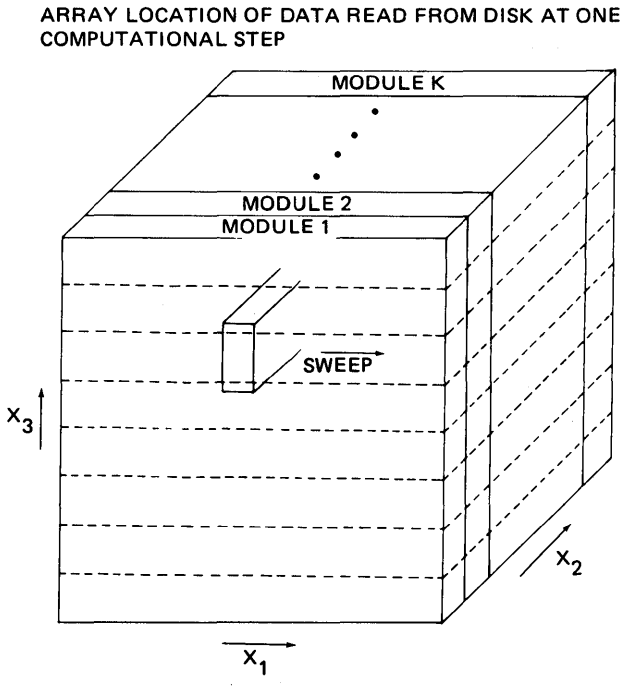


Figure 3—Array location of data read from disk at one computational step

here, but the $\rho^{(n)}$ data is first Fourier transformed in the x_2 direction before being written onto disk (as $\tilde{\rho}^{(n)}$).

The third phase involves taking x_1 and x_3 transforms of the $\tilde{\rho}^{(n)}$ array, dividing each element by a (constant) function of position, and taking inverse x_1 and x_3 transforms. The Poisson solution is complete when the transformed variables (now denoted $\tilde{P}^{(n)}$) are transformed in the x_2 direction at each step as they are read (in the first phase of the next time step). In the third phase, essentially, a transpose must be done so that entire x_1-x_3 planes of data can be brought into random access memory.

There is an extra degree of freedom in our disk addressing that was not used in the first two phases, but which will be important here: The disk has four separately addressable surfaces. Thus, if data corresponding to all x_2 values is properly arranged on the four surfaces, then we can read a set of entire x_1-x_3 planes corresponding to $^1/_4$ of the different x_2 values by reading one surface. This can be done at close to contiguous-block rates: where all the variables that pass under the one recording head are used. The total reading (or writing) time is essentially the same whether we are reading $^1/_4$ of the $\tilde{\rho}^{(n)}$ array, or any smaller fraction. For a system with 96K 16 bit words of random access memory, we can store $^3/_{16}$ of the array and essentially read and write the variables at $^3/_4$ contiguous-block rate. The total number of variables of each type transferred to and from disk together with the total number of accesses and timing estimates are given for the three phases in Table I.

Microcomputer operation

The microcomputer modules each have an independent instruction, data and scratchpad memory. Most of the random access memory resides in the microcomputer data memories. A simple control scheme that we are currently implementing on the prototype system (with two modules) involves a time-multiplexed bus attached to each microcomputer. The host (minicomputer) selects each module in turn by enabling its output buffers. The selected module transfers data to and from the host by DMA. When the transfer is complete, based on a signal from the module, the host selects the next one and the module starts computing.

The arithmetic operation count for the three phases is given in Table II in terms of the number of (one-dimensional) FFT's, square roots, divisions and other additions and multiplications. The total computing time estimates for one module are also given assuming characteristics similar to the

TABLE I.—Disk Transfers

PHASE	VARIABLE	ACCESSES	WORDS READ	WORDS WRITTEN	TIME
1	$\tilde{u}_i$	152	2.9×10^6	1.8×10^6	62 sec (26)
	$\tilde{p}$	160	1.3	0	23 (13)
	v_i	128	1.5	1.5	42 (18)
	TOTAL	340	5.7	3.3	127 (57)
2	v_i	72	2.1	0	28 (12)
	$\tilde{p}$	32	0	.5	8 (4)
	TOTAL	104	2.1	.5	36 (16)
3	$\tilde{p}$	6	.8	0	9 (3)
	$\tilde{p}$	6	0	.8	9 (3)
	TOTAL	12	.8	.8	18 (6)

*( ) REFERS TO FAST DISK

TABLE II.—Computations

PHASE	FFT	SQRT.	DIV.	MULT.	ADD.	TOTAL*
1	4×10^3 (REAL 64 PT.)	$.25\times10^6$	0	12×10^6	50×10^6	245 SEC.
2	4×10^3 (REAL 64 PT.)	0	0	$.25\times10^6$	6×10^6	32 SEC.
3	8×10^3 (COMPLEX 64 PT.)	0	$.25\times10^6$	0	0	90 SEC.

*TIME FOR ONE 16 BIT MODULE

ones used in our system. It can be seen that the throughput of ~2 modules will approximately match the disk transfer rate for our prototype system, and ~26 modules for a system with a faster disk.

A simple module data allocation scheme, to be used in our system, involves partitioning the data set at each step so that each module has data corresponding to the same x_1, x_3 values but different sets of x_2 values (with the exception of the Fourier transforms in the x_2 direction, which will be done separately in each module.) As in the disk allocation scheme, some $\tilde{u}_i^{(n)}$ and $P^{(n)}$ data must be accessed twice, but from the host instead of the disk and by two separate modules. This redundant data set is directly proportional to the number of modules used, and is approximately 7 K (16 bit) words per extra module. A 4 K buffer space is also necessary for each module. Thus, the total memory requirements increase by ~11K for each module added. For systems with more than several modules, an alternative to redundant storage would be to use inter-module communication. The time-shared bus will become saturated if used with more than ~6 modules. Beyond this number, a ring network with direct intermodule transfers would be a very efficient scheme.

Assuming buffered operation, the total time for one time step will be about 210 seconds for our system with 2 modules and 90 seconds for a system with a faster available disk and 6 modules. The time quoted for a similar algorithm (coded in Fortran) on a CDC 7600 is 180 seconds per time step. An assembly language coded version should run several times faster. Between 50 and 200 time steps are required to resolve a significant portion of turbulent kinetic energy decay in many of these simulations, resulting in a total solution time of several hours per case.

A straightforward way to double the performance of the system would be to use two disks and either 12 16 bit modules or 6 expanded 32 bit ones. Such a system, with 300 megabyte disks would be able to solve $(256)^3$ problems, with 200 time steps taking ~5 days; a reasonable time for a moderately priced, dedicated system.

MULTI-MODULE SYSTEMS

We looked at ways to increase the solution speed for our problem by utilizing more modules, and at ways to increase the grid size (data base) by utilizing more disks. It turned out that a very simple configuration consisting of 8 of the systems described above could be used to give a speed increase of ~7 and a data base increase of 8. Our reason for looking at larger systems was mainly to give an example of a highly parallel implementation of a particular flow problem. If other types of flow problems prove to be amenable to this type of architecture, or if the system cost decreases enough in the future, it may be worthwhile to build such a system.

We define a *unit* to be a minicomputer, set of microcomputers (16 or 32 bits) and one or two disks as described above. The configuration studied is quite simple: we just have to dual-port each disk and form a ring configuration (see Figure 4). For the $(64)^3$ problem, 8 sweeps (in the x_1 direction) were required for a single unit for the first two phases of each time step. We could then use 8 units—one for each sweep. With some minor exceptions, each module could read the same variables (during a sweep) as if it were doing the entire problem. These variables would be available on the two disks (or four disks) connected to each minicomputer. If the disk transfers were coordinated so that each left-hand disk were accessed at the same time by each minicomputer and then each right-hand disk, there would be no disk conflicts. This synchronization would require the testing of a flag only after the transfer of several hundred variables, and could be done using straightforward software techniques and standard interprocessor busses (IPB's). An advantage of this scheme is that one unit by itself could use essentially the same program for these phases and solve the entire problem. Also, by changing the (x_3) width of the sweep of each unit and the number of sweeps each unit must take, we could conveniently use any number of units up to 8. This should simplify development of the full system and decrease time lost due to failures. The computational speed should increase roughly linearly (in these phases) with the number of units.

We now describe a scheme for the third phase. For the single unit, approximately 10 percent of the disk transfer time was associated with this phase. With 8 units, we can segment the data ($\tilde{p}$ array) in the x_2 direction into 8 groups and rearrange the groups among the units. Each unit then would have to transfer 7 groups to the other 7 units. By transferring around the ring first in one direction then in the other, the rearrangement can be done in seven steps. During the operation, each piece of data moves an average of 2 units and we finally have 8 entire $(x_1\text{-}x_3)$ planes of data in

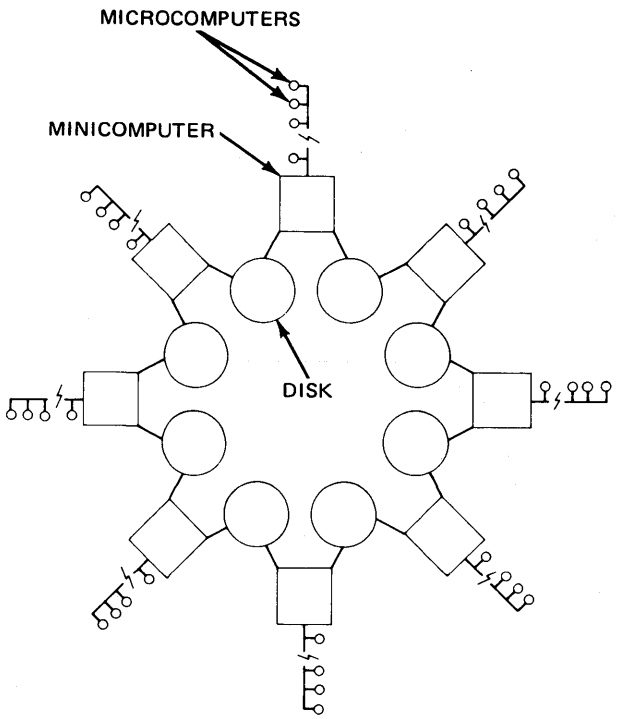


Figure 4—Multi-module system

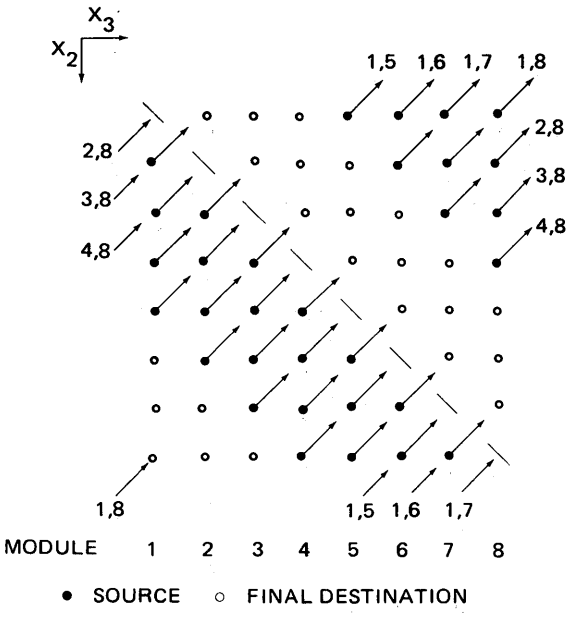


Figure 5—Transpose—First step

each unit (corresponding to different x_2 values). The flow of the data (in one direction) is depicted in Figure 5. Each solid line represents a transfer at the first step. The number of groups transferred by each unit is 4, 3, 2 and 1 for the respective right-steps, and 3, 2 and 1 for the left-steps. (Most of these transfers can be overlapped with the first two phases.) The entire phase should take about $^3/_8$ as long as for a single unit. To complete the phase, we then have to do a two-dimensional (x_1-x_3) transform in each unit, a division, an inverse transform, and then an inter-unit transfer again. The time required for the total time step (all three phases) is ~6 sec., compared to 44 sec. for one (dual disk) unit, giving a speedup of ~7.

For a $(256)^3$ array, we have 640 and 320 seconds per time step, respectively, for single and dual disk systems. For 200 time steps, this gives ~40 and 20 hours. Dual 300 Mbyte disks and 6 32 bit microcomputers would allow a $(512)^3$ solution (200-400 time steps) in 6-12 days.

There are some advantages to using the system that we have described. First, since it is a one-dimensional configuration with simple connections, adding new units or disconnecting failed ones should be straightforward, both in software and hardware. This also applies to each unit, which is a linear configuration of microcomputers. Second, although we have 48-96 (32 or 16 bit) microcomputers operating in parallel, only groups of 6-12 microcomputers need be coordinated, and only a group of 8 units need be coordinated. Third, each unit can have its own operating system and do an entire problem by itself, with any number of microcomputers.

This type of simple arrangement is very efficient for all the operations required except the one-dimensional Fourier transforms in the x_3 direction. Since a relatively small amount of data is associated with this operation, the overall system is efficient.

DISCUSSION

We have described a series of machines, assembled from available subsystems, that can have a throughput ranging from a fraction of that of the CDC 7600 to an order of magnitude greater than the 7600 for a specific time dependent three dimensional flow problem. The cost of the machines range from approximately $50,000 to over $500,000, and, for each, the cost for a solution to the problem studied is almost two orders of magnitude less than for the 7600. In these machines, extensive use of buffering and software synchronization can be made so that inter-component connections are very simple. Also, the systems can be highly modular, with more powerful machines consisting of sets of simpler machines connected together. Thus, a program to develop machines along these lines and gradually increase complexity and computational speed should be fairly straightforward.

The systems use sets of microcomputers to do most of the computations, and the problem is decomposed into parts to be solved independently by each microcomputer, as if it were doing the problem sequentially. This "macroparallel" approach should be compared to other approaches, such as one where sets of tightly coupled functional units operate concurrently on a single data stream. The AP8-120B peripheral processor is the most popular version of this.

We have described specialized algorithms to solve our problem on the machines considered. The main part of these algorithms concerns data flow. We found that special attention must be paid to this if effective use is to be made of the moving head disks. Use of these disks will be necessary, at least over the next few years, for economical storage of very large data bases in modest systems.

The problem studied represents a specific type of computational method (two time level, explicit) and a particular set of boundary conditions (periodic). Other methods (e.g., implicit) and boundary conditions should also be considered. Like our method, versions of implicit methods have semi-local and global classes of computations. Also, these global classes often involve solutions of one-dimensional tridiagonal equations in three independent directions, and a comparison with our study would be interesting.

Finally, software will, of course, be very important if wider classes of problems are to be solved. Since the microcomputers operate sequentially, higher level language implementation should not be too difficult, although efficient use of the small high speed control stores may prove otherwise. Another difficulty appears to be in the efficient use of the disk. Perhaps classes of data flow algorithms can be coded and used with a higher level language.

REFERENCES

1. Kottler, C. and R. McGill, "The Feasibility of Using Minicomputers for Reducing Large Problem Solving Costs," *Instruments and Control Systems*, Vol. 46, 1973, p. 57.
2. Orszag, S. A., "Minicomputers vs. Supercomputers: A Study in Cost Effectiveness for Large Numerical Simulation Programs," Flow Research Note No. 38, Flow Research Inc., 1973.

3. Allen, J., "Computer Architecture for Signal Processing," *IEEE Proceedings,* 1975, p. 624.
4. Karplus, W. J., "Peripheral Processors for High Speed Simulations," *Simulation,* Vol. 29, 1977, p. 143.
5. Mansour, N. N., P. Moin, W. C. Reynolds and J. H. Ferziger, "Improved Methods for Large-Eddy Simulations of Turbulence," *Proceedings Symposium on Turbulent Shear Flows,* University Park, Pa., April 1977, p. 14.21.
6. Clark, R. A., J. H. Ferziger and W. C. Reynolds, "Evaluation of Subgrid-Scale Turbulence Models Using a Fully Simulated Turbulent Flow," Report No. TF-9, Department of Mechanical Engineering, Stanford University, 1977.
7. Roache, P. J., *Computational Fluid Dynamics,* Hermosa Publishers, Albuquerque, N.M., 1972.

Applications of minicomputers to Library Systems and Services

SESSION CHAIRMAN—LAURA J. RAINEY
Rockwell International

Panel Members

Audrey N. Grosch—University of Minnesota
Charles M. Goldstein—National Library of Medicine
Karl M. Pearson, Jr.—California Library Authority for Systems and Services

SELECTING A MINICOMPUTER SYSTEM—Audrey N. Grosch

Factors influencing the choice of minicomputer systems for library automation will be discussed. Specific conditions to be treated will be:

Whether already tailored, packaged systems provide the necessary functional capability;
Whether program development and hardware systems will be configured and produced by one system vendor or by custom programming by a firm specializing in programming for a given manufacturer's smaller or larger configurations;
Whether the library has in-house capability for hardware/software evaluation as opposed to using contractors or consultants for this choice;
Whether total program development for applications will be done using commercially available DBMS (Data Base Management System) software packages such as IMAGE, MUMPS, DBMS-11;
Whether parent organization has standardized toward one manufacturer for machines to be used in connected mode to a large main frame computer;
Whether the system will be a stand alone or remote link to a main frame computer or a shared use of a larger minicomputer configuration having a multi-user, multi-tasking operating system.

Some characteristics of minicomputers presently available, together with a description of the range of peripheral devices will also be presented along with some procurement guidelines.

LIBRARY AUTOMATION PROGRAM AT THE LISTER HILL NATIONAL CENTER FOR BIOMEDICAL COMMUNICATION—Charles M. Goldstein

Lister Hill National Center for Biomedical Communication at the National Library of Medicine is supporting an R & D program in library automation. One aim of this program is to develop an integrated minicomputer based library system to satisfy the following objectives:

Transportability
Expandability
Multiple levels of on-line user
interbase and systems network access

Transportability refers to the ability to implement the system on more than one type of computer as well as easily maintain the program, to wit, a higher order language implementation. Expandability is defined as the ability to respond to increased loads by scaling up the equipment without changes in the program. Systems network access implies automatic access to network resources by the system without manual intervention on the part of the user. An example of the latter would be automatic access to a network available catalog file, if an item is not found in a local file.

The initial modules under development are circulation and acquisition modules. The potential impact of such systems on small, medium, and large libraries, as well as their potential impact on a National Library Network architecture will be discussed.

MINICOMPUTERS IN LIBRARY NETWORKING—Karl M. Pearson, Jr.

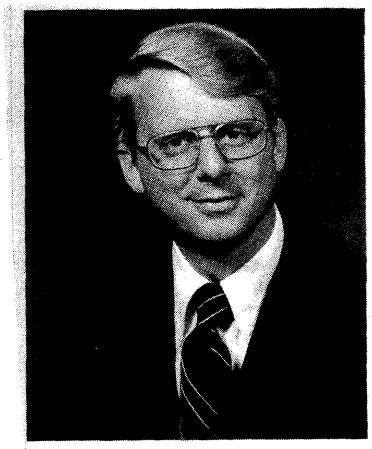
Shortly after minicomputers appeared in library applications as the base for turnkey circulation control systems, their value for networking applications was realized. First, they have been put to use as front-end processors and communication controllers for the centralized, on-line catalog systems, most notable, OCLC and BALLOTS. Next, they

have begun appearing in regional service centers. The New England Library Network (NELINET) has installed a minicomputer as an interface between member libraries and OCLC. In addition to serving as a data concentrator and communications handler, experiments have been run using the minicomputer as a sophisticated message store and forwarding device to facilitate interlibrary loan.

Several libraries have procured minicomputers for general in-house data processing needs, particularly for economical text editing, data checking, and on-line file maintenance. Many of these computers also serve in a distributed processing role in which they can link to a host maxicomputer for large-scale processing requiring extended sorting and formating of large bibliographic records.

Future developments are likely to see a minicomputer based communication network to pass bibliographic data from one utility to another, (e.g., between OCLC and the Library of Congress), to transmit interlibrary loan transactions, and allow user access to any library data base regardless of the type of terminal being used.

PART II—METHODOLOGY



Area Director:
Stephen R. Kimbleton
National Bureau of Standards
Washington, DC

Performance measurement and evaluation

Computer systems performance analysis has matured rapidly within the last decade. Ten years ago, the field was just beginning to gain prominence. Results were sparse and general methodologies were lacking. The efforts of individual researchers were exciting but difficult to interrelate.

The present state-of-the-art performance is a dramatic improvement over that of ten years ago. Although the performance problem cannot be regarded as solved, a very significant degree of progress has been made. General purpose tools are transitioning from the feasibility investigation stage to the prototype stage. Major components of a possible structure for the field can be discerned and are reflected in the following discussion.

The four performance analysis sessions at the 1978 National Computer Conference were structured to provide a unique opportunity to learn about some of the most exciting current developments in the field. The presentations and their associated papers describe some surprisingly sophisticated tools which are currently at the prototype stage together with a context for their appreciation.

NCC78 performance sessions covered a spectrum of issues from enduser and installation manager requirements to theoretical developments which are the prerequisite to future implementation of even more sophisticated tools. To accomplish this, the sessions were structured to focus on: Computer Performance Management, Computer Performance Technology, Computer Performance Modeling, Mathematical Analysis of Computer Performance and End User Performance Considerations.

COMPUTER PERFORMANCE MANAGEMENT

Evaluating performance from the viewpoint of an enduser or installation manager is complicated by two factors. The first is the immense volume of performance statistics which can be collected using current hardware or software monitors. The second is the problem of handling the collected information. The Computer Performance Management session, chaired by Phil Kiviat, focused on these two problems.

Solving the first problem requires prior determination of those performance measures which are relevant for the needs of endusers and installation managers. In contrast with other areas of science and technology, the need is not for more

data but for the right data. Presentations and the associated papers concerned with this issue covered both the types of measures which are useful and their relationship to the dependence between data processing staff and data processing users.

Given that the desired data has been identified, the next requirement is automating its handling. This requires a management information system for performance data. Major issues which must be considered in the development of Performance Management Systems are described in the context of recent work in this area.

COMPUTER PERFORMANCE TECHNOLOGY

Unacceptable system performance requires changing either the system or its workload. Provided that the system and its workload can be appropriately characterized, performance prediction techniques can be used to evaluate the likely results of specific changes and to answer "what if" questions.

The Computer Performance Technology session, chaired by Steve Kimbleton, described new techniques for characterizing the workload of a computer system through classifying it into clusters. In addition, a new approach for characterizing individual jobs was described. Taken together, these two characterizations provide a more effective means for developing the required input information for performance prediction tools. A fast, user oriented queuing network based performance prediction technique was also described.

COMPUTER PERFORMANCE MODELING

Modeling plays a key role in evaluating "what if" types of questions. Model development can be pursued via either analytic or simulation based approaches. Selection of either approach usually forces a tradeoff between the design insight provided by analytic models and the level of detail which can be reflected in simulation models. As a result, careful consideration of the goals of the model user is required.

The Computer Performance Modeling session, chaired by Jeff Buzen, described the requirements of an installation manager for effectively using modeling techniques. A new hybrid approach based on combining simulation and analytic techniques thereby permitting both increased levels of detail and faster speed was also described. In addition, a model for investigating peripheral processor performance was also presented.

MATHEMATICAL ANALYSIS OF COMPUTER PERFORMANCE

Computer system models are a key source of design insight. Developing analytic forms of such models requires close interaction between researchers in the areas of queuing and other types of stochastic processes and model oriented performance analysts.

The Mathematical Analysis of Computer Performance session, chaired by Hisashi Kobayashi, described some recent results in the interdisciplinary area of systems modeling. New techniques for evaluating the performance of queuing systems were presented. These results included a new entropy based approach to obtaining the time dependent behavior of certain queues and the distribution of the cycle time of a closed queuing network. In addition, new results in evaluating the performance of processor scheduling algorithms were also described.

END USER PERFORMANCE CONSIDERATIONS

System performance, as viewed by an end user, can be improved by enhancing the performance associated with a given program or, alternatively, by simplifying the user's interaction with the system. This session chaired by Shirley Watkins, is concerned with aiding the end user in efficiently using computer resources through simplifying their use or evaluation. The first paper will describe a technique for faster and simpler program design verification through using a data generator for program testing.

The second paper looks at the end user-system relationship in the context of providing an adaptive system interface. These formal presentations will be complemented by an informal panel discussion.

How to improve your performance through obfuscatory measurement*

by DAVID F. STEVENS
University of California
Berkeley, California

INTRODUCTION

It is best, I believe, to make it clear at the outset that this monograph has a somewhat obfuscatory title . . . for it is not my intent actually to show you how to improve your performance (that being a merely technical exercise devoid of true intellectual challenge), but rather to show you how to claim—and substantiate—superlative performance, even in the face of extreme user discontent.

As I have noted elsewhere,[1] there are two basic elements of success in computer center management: the achievement of saturation and the demonstration of efficiency. That earlier paper was somewhat deficient in that it concentrated upon the one (achievement of saturation, independently of the true—users'—workload) and slighted the other; it is hoped that this effort will somewhat redress the balance.

At this point I should, in all modesty, note that none of the techniques discussed here are my own creation. My small contribution consists only in the recognition of these scattered efforts as elements of a significant whole. I should be quite surprised, in fact, if all who read this treatise have not already either employed or encountered every measure discussed herein. What has heretofore been lacking is a true appreciation of the difference between a system viewed through the rose-colored glasses provided by these measures, and that same system viewed in the cold light of reality. It is my belief that the craftsman who understands his tools uses them most effectively; it is my hope that, after having read this paper, you will have a more complete understanding of obfuscatory measurement: what it is, what it can do for you, and, perhaps, how to devise your own obfuscatory measures. To that end, then, I will define what I mean by "obfuscatory measurement", provide some general rules and principles for the creation of obfuscatory measures, and give a number of examples to demonstrate the power of obfuscation and hint at the astonishing variety of forms it may take.

* This work was done with support from the United States Department of Energy. The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-7405-ENG-48. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

OBFUSCATORY MEASUREMENT DEFINED

"Obfuscatory measurement" is measurement which obscures that which it should illuminate.

To be quite precise, *any* measurement which gives a false impression is obfuscatory; we are here concerned, however, only with those which allow DP Management to paint a brighter picture than that seen by the users, for it is only those which contribute to their—the DP Managers'—success. Although you may be unfamiliar with the term, you will discover that you are familiar with many of the measures . . . for many (if not, indeed, most) of the performance measures in common use today are obfuscatory. Succinctly stated, obfuscatory measures

- measure the wrong things
- measure the right things . . . wrongly
- measure something else (i.e. other than that which they purport to measure)
- measure nothing at all (or at least no meaningful thing)

Furthermore, since systems, like people, respond to measures by which they are evaluated, these measures contribute to success (as defined in Reference 1), by rewarding pessimal performance, thus increasing saturation.

GENERAL RULES, PRINCIPLES, AND TECHNIQUES

1. Select your measures with care—Not all measures are appropriate to all situations. You should neither attack the fly with the cannon, nor the elephant with the feather-duster. Tailor your measures to the tractability of your users and the gullibility of your upper management . . . and always have a couple in reserve, just in case. . . .
2. When in doubt, seek the advice of your mainframe vendor—Remember, your vendor cannot sell you additional equipment until your upper management is convinced of the saturation and effective utilization of your existing configuration. Furthermore, your vendor has a wealth of experience in dealing with upper managements just like yours. . . . Obfuscation is the very

essence of the salesperson's art; as you seek legal advice from a lawyer, you should seek obfuscatory advice from your vendor.

3. The easier a measure is to obtain, the more likely it is to be obfuscatory—This is one of the few cases known to modern science where Murphy's Law operates in favor of the practitioner. (There is no great mystery here, however: it is often the generosity of the vendor which made the measure easy to obtain.)

   Two specific kinds of easy measure are worthy of individual mention: *means* and *percentages*. As we shall see in the discussion of Availability, suitable definition of the base can turn any measurement into a praiseworthy percentage. As for the mean, while it frequently lacks meaning, it often exhibits meanness. Even though a number of articles in the recent literature (notably a pair by Gary Carlson[2] and one by Jain[3] have emphasized the obfuscatory nature of "indiscriminate" use of statistical concepts, the mean is so beloved by the average person that its utility is expected to continue relatively undiminished. One can still, for instance, report a favorable mean in preference to a realistic median in most circumstances. It is also often fruitful to ignore the distribution. More details on this technique may be found in Reference 4.

4. Overextend analogies—Concepts which are meaningful in other fields can sometimes be transferred into the computer performance arena, where they are invalid, without loss of prestige. It helps, of course, if the concept is so familiar that it is accepted without question in its new context. MTBI is such a measure.

5. Creative Definition—This is an indispensable element of the obfuscator's arsenal . . . for the most misleading percentage you can devise won't help you unless you can convince someone that it measures *something*. If yours is an elementary situation, actual definition is not important: a catchy name is all that is required . . . (Remember "CPU efficiency"? was there anything efficient about it? A modern example is "depth of multiprogramming".)

   If you find yourself in deeper waters, some measure of definition must be supplied . . . but it is best if it is either ambiguous or incompletely specified. ("Availability" as "percentage of time available" is, as we shall see, an excellent example of this technique.)

OBFUSCATION IN PRACTICE—SOME SPECIFIC EXAMPLES AT WORK

Some of these measures, most particularly Availability and MTBI, have been discussed extensively elsewhere;[1,4] inasmuch as they are among the most widely employed obfuscatory measures, however, they are included here—briefly—for completeness. We will look at four general classes of measures, giving at least two examples of each. This should provide an ample foundation upon which you can construct the obfuscatory program most suited to your specific situation. (The contents of this section are summarized in Exhibit I for quick reference, together with a listing of comparable honest measures—which are offered in the belief that it is as desirable to know your enemy as to know your friends.)

1. Measures of Availability and Reliability
   (a) Availability

   Usually expressed as a percentage, "availability" is taken by the uninitiated to indicate the amount of time the system is usable, whereas in fact it indicates the amount of *scheduled* time the system is available *to the computer center*. By reducing the base to scheduled time a significant increase in percentage is obtained. It is further increased by including many periods of time when it is not, in fact, fully usable: start-up times, time spent re-running lost or interrupted jobs, and time devoted to the "run-down" before a scheduled interruption. Exhibit II illustrates the cumulative effect of all these adjustments. It shows a week in the life of a one-shift operation, with one period of preventive maintenance (PM), a daily system development shot (SD), two unscheduled periods of down-time (15 minutes on Tuesday and an hour on Friday); start-up requires half an hour, and "run-down" starts a half-hour before system development time and an hour before the end of the shift. Naive and obfuscatory measures stand in rather sharp contrast.

   (b) MTBI (Mean Time Between Interruptions)

   MTBF (the mean time between failures) is so well accepted as a reliability measure in engineering contexts that practically no one questions its DP analog, MTBI. That the causes of failure in the two fields are largely unrelated is largely ignored: failures in mechanical systems are caused by wear and fatigue (to which software is impervious); failures in computing systems are caused by unexpected input (to which mechanical systems are rarely exposed) and trivial overflows (which, if they cause damage at all, cause trivial damage: will an overflow on the meter crash a taxi?). The user-oriented measure which most closely corresponds to MTBI is the mean (or median) service interval. To see how they compare, we return to the sample week of Exhibit II. The mean service interval, even giving full credit for the run-down periods, is 2.23 hours (26.75/12), and the median is 2.5. The *conservative* way to calculate MTBI is to divide "hours available" by "number of interruptions plus 1": 32.75/3=10.9 hours . . . more than three times as long as the *longest* service interval.

2. Measures of Throughput
   (a) Utilization

   When the obfuscator is asked for measures of throughput she has ample industry precedent for responding with measures of utilization. Utilization measures are advantageous because they reward ineffective programming (which is much easier to

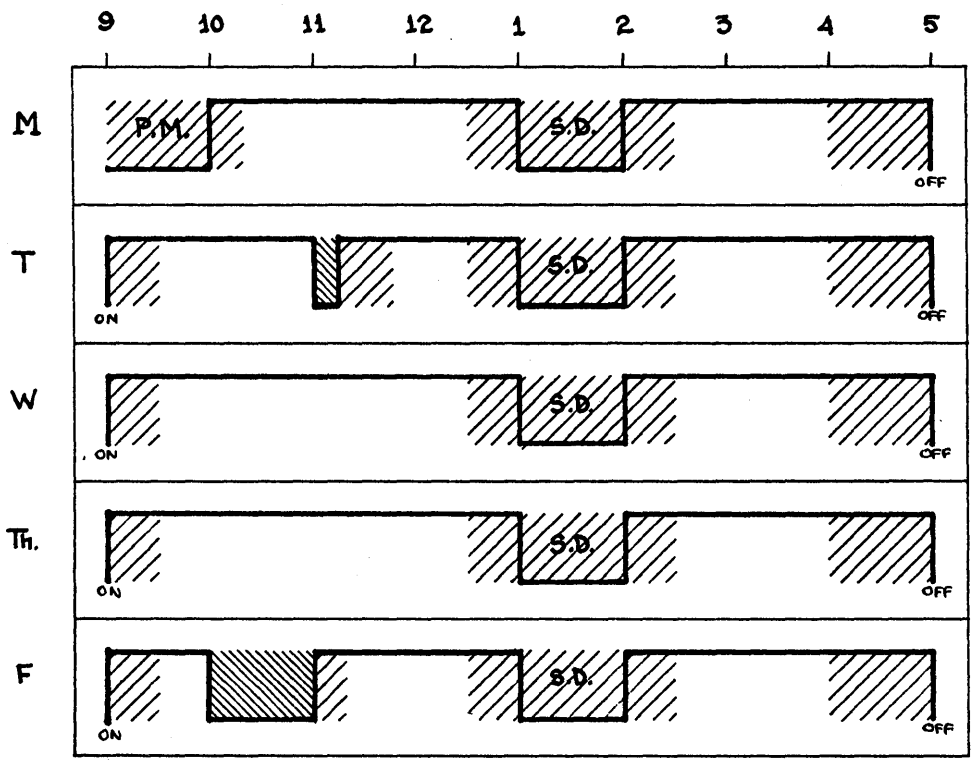
Class	Honest Measures	Sample Obfuscatory Measures (and what they actually measure)
Availability and Reliability	True availability to the users. Mean and median service interval.	Availability (% uptime) MTBI (mean scheduled time to crash)
Throughput	Work delivered (in user units).	Utilization (resource occupancy) Efficiency (utilization)
Turnaround	Processing time vs. system wait time, by category.	Interactive response time Turnaround time (as long as you stick to means, they measure nothing meaningful)
General Productivity	Concurrency: the number of CP and channel activities occurring simultaneously. Existence (or not) of saturation. Capacity relative to workload. Functions delivered; quality of code.	Overlap (existence of overlap) Depth of multiprogramming (number of active initiators) Saturation (work, productive or not, as % of "capacity") Lines of code (prolixity)

Exhibit I

obtain than the other kind). Your path here is not quite as free as it used to be, what with the introduction of distinguishable "system" and "problem" states for CPU utilization . . . but it remains the case (thanks to your friendly mainframe vendor) that much of what is called "problem state" is actually system overhead. And it seems extremely unlikely that anyone is going to come up with a meter which distinguishes between "system" and "problem" channel activity states!

(b) Efficiency

This is actually a vestige of the past, but one which has validity in some contexts, and adds a certain panache to your reports. It may be freely used in place of "utilization" (the two are identical in meaning), but I would advise against using them both to refer to the same quantity: such a juxtaposition might inspire tiresome questions. "CPU utilization" and "channel efficiency", on the other hand, provide a nice appearance of breadth.



D.P. MGMT. (AVAIL/SCHED)	USER (AVAIL/TOTAL)
6/6	3.5/8
6.75/7	3.75/8
7/7	4.5/8
7/7	4.5/8
6/7	3/8
32.75/34	19.25/40
96%	48%

Exhibit II

3. Measures of Turnaround
   (a) (Interactive) Response Time
   This has superseded "turnaround time" as the most commonly quoted measure of turnaround, but the principles of use are the same. It is most important here to remember that the mean can be manipulated by stacking the extremes. To be specific, you can achieve essentially any response time you wish by requiring a suitable number of trivial interchanges—with zero response time—to take place during any interactive session. (This is easily seen to be the inverse of the situation shown in Exhibit III, wherein a Miraculous improvement in "mean service interval" is achieved by introducing an unused—and hence uninterrupted—weekend service.)

   Another fact to be borne in mind is that, in some situations, response which is *too* quick creates tension, which causes errors . . . and errors lead to wasted work, thus bringing saturation (and hence success) ever closer. (A better strategy, however, is to strive for consistently unexpected response time, whether it be quicker or slower than anticipated . . . but this is somewhat off the subject of this paper.)

   (b) Turnaround Time
   Since the good turnaround times are the small ones, this is a situation where the median, surprisingly enough, favors the obfuscator. Nevertheless, I recommend sticking with the mean. For not only is the median a dangerous precedent to set, the mean is, as we have seen above, quite a tractable index. As in the case of response time the enterprising manager can cause enough small jobs to be submitted to achieve whatever mean turnaround time is deemed necessary.

4. Measures of General Productivity
   (a) Overlap; Depth of Multiprogramming
   These two measures are grouped together not because they are thought to be equivalent (they are not), but because they address the same problem: a vague understanding on the part of upper management that some multiplicity of processing is desirable. They make a good combination, not only because they obfuscate in different ways, but also

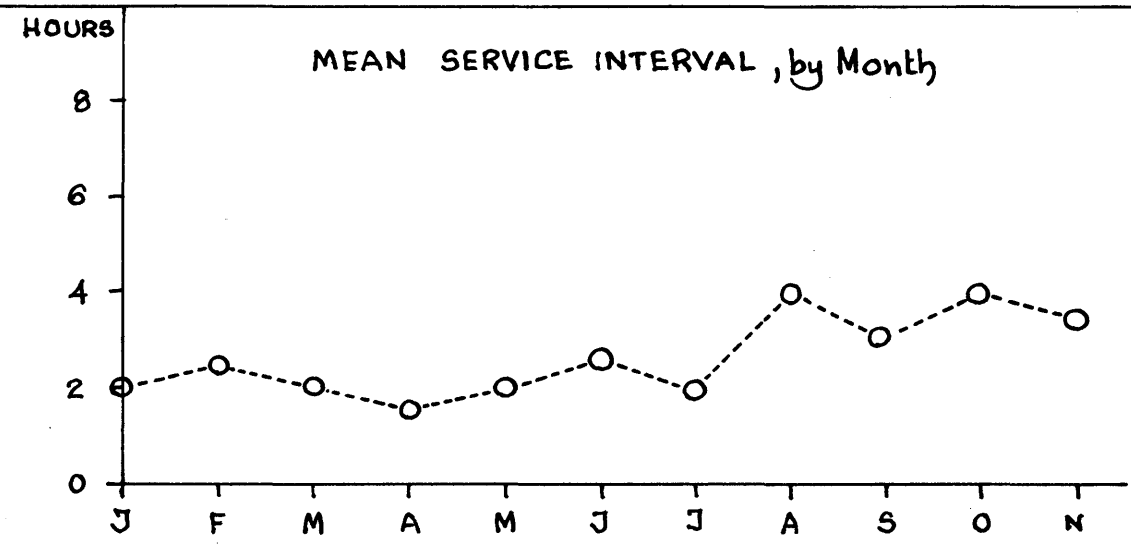


EXHIBIT 3a.

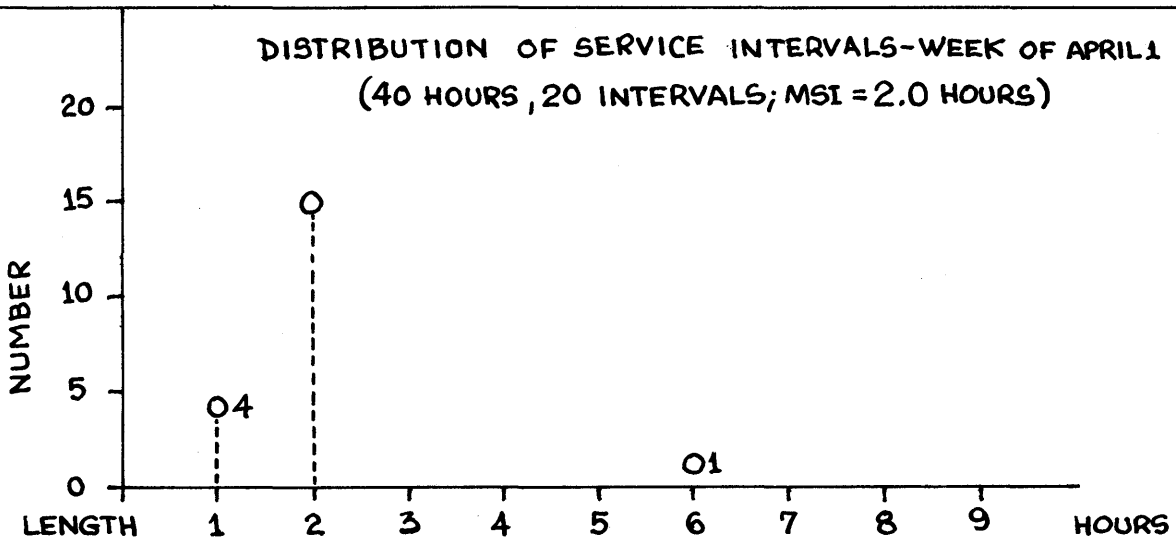


EXHIBIT 3b.

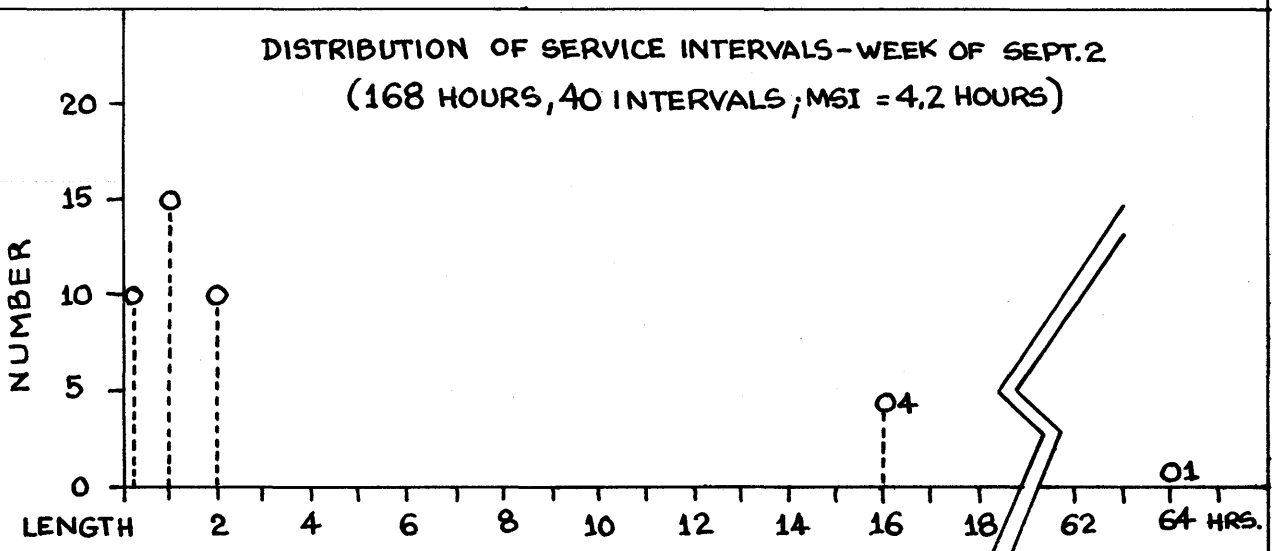


EXHIBIT 3c.

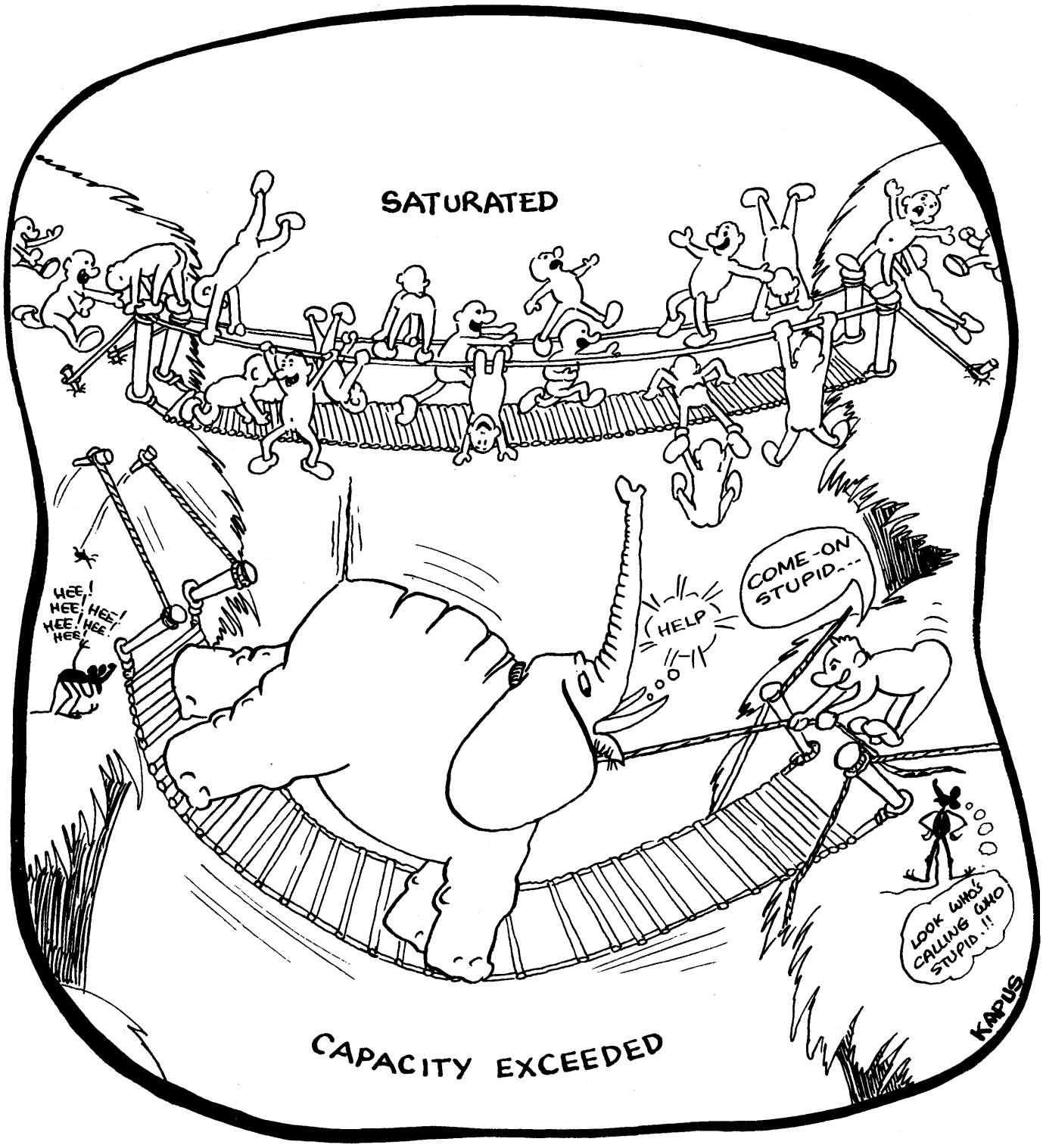

SATURATED
HEE! HEE! HEE! HEE! HEE! HEE!
HELP
COME-ON STUPID---
LOOK WHO'S CALLING WHO STUPID-!!
CAPACITY EXCEEDED
KAPUS

Exhibit IV

because the two together give no more accurate a picture than either one singly.

Overlap is in fact somewhat less obfuscatory than depth of multiprogramming, for it measures the percentage of time that some amount of simultaneity is experienced; it does not, however, consider the level of simultaneity. (Thus two simultaneous processes are every bit as good as seven.) It may be this very touch of honesty, paradoxically, which makes overlap so useful as an obfuscatory measure.

Depth of multiprogramming, on the other hand, is pure obfuscation: it counts initiators instead of processes. In many shops, large values of depth of multiprogramming survive as tribute to the memory salesperson's art, while all the jobs lie quiescent awaiting the pleasure of the Resource Manager or some other such system magus.

(b) Saturation

The obfuscatory nature of "saturation" lies in the fact that saturation is not a measure but a binary condition: the change in the quality of a service which moves from an unsaturated condition to a saturated one is an abrupt discontinuity: service effectively stops and the input queue becomes infinite. References to "80 percent of saturation" thus really mean "80 percent of capacity", and are doubly obfuscatory because saturation can be largely independent of capacity. We can best illustrate this by means of an example: Consider a long, narrow footbridge with a capacity of 2000 pounds. Thirty-nine small boys, each weighing less than 50 pounds, would not exhaust its capacity but would surely saturate it (they wouldn't all fit at once) for quite a while. A single man leading an elephant, on the other hand, would not saturate the bridge but would exceed its capacity (see Exhibit IV).

A useful related obfuscation is the measurement of throughput as a percentage of capacity, for many people ignore the basic fact that "capacity" changes with workload and environment. They consider "capacity" a configuration constant, despite the fact that they know full well that any reasonable multiprogramming system has a smaller capacity when restricted to highly compute-bound jobs than when fed a mixture of compute- and I/O-bound work. This blind spot can be exploited in other ways; it is much less well-known, for example, that any multiprogramming system strongly dominated by priority considerations has a smaller capacity than a system free to assign requested resources (such as the CPU) in an optimal fashion.[5] A word to the wise, one hopes, is sufficient. . . .

(c) Lines of Code

This measure, being directed at human productivity, might be considered by some to be somewhat outside the scope of this paper, but programmer performance *is* an element of computer center performance and lines-of-code *is* superbly obfuscatory. More importantly, lines-of-code is perhaps the best embodiment of one of the great philosophical rocks upon which Obfuscatory Measurement is founded:

> It is nearly always possible to substitute numbers for judgment.

A timid person might hesitate to use lines-of-code on the grounds that it is patently absurd (is the Beer Bottle Song ["One hundred bottles of beer on the wall. . . ."] better than a Shakesphere sonnet? a limerick than a haiku? this paper than the Gettysburg Address?), inasmuch as it ignores quality. Such a person severely underestimates the power of numbers to convince and confuse . . . in a word, to *obfuscate*.

L'ENVOI

It must, alas, be admitted that an occasional obfuscator is taken to task for an excess of creative zeal. Should that unhappy fate befall you I can offer no defense better than that provided by Pooh Bah, who under such like circumstances,[6] claimed that his obfuscation was "merely corroborative detail, intended to add artistic verisimilitude to an otherwise bald and unconvincing narrative."

REFERENCES

1. Stevens, D. F., *The Computer Managers Guide, Datamation,* June 1976.
2. Carlson, G., *The Use of Statistics in Performance Measurement,* Parts I and II, *EDP Performance Review,* August and September 1977.
3. Jain, A. K., *Statistical Approaches in Computer Performance Evaluation Studies,* Tutorial at CPEUG 13, November 1977.
4. Stevens, D. F., *Obfuscatory Management, SIGMETRICS/CMG VIII Proceedings,* November 1977.
5. Stevens, D. F., *On Overcoming High-Priority Paralysis in Multiprogramming Systems,* CACM, August, 1968.
6. Gilbert, W. S., *Mikado,* Act II, March 1885.

Perceptions of performance

by RICHARD J. CIESLOWSKI
Aetna Life and Casualty
Hartford, Connecticut

INTRODUCTION

Most businesses view the data processing staff as a centralized resource, providing a specialized service to all segments of an organization. Centralization has occurred in response to the instrinsic cost and expertise benefits available through a concentration on this discipline. The inevitable separation from the client organization results in the establishment of a consultant/client relationship, e.g., the personnel responsible to support and accomplish the organization's operational objectives reside in a separately managed and controlled area. This presents a significant problem—one which often results in the operational area being accountable for an objective, while the data processor holds the key to its accomplishment. The data processor, who is charged with maintaining a high level of technical data processing competence along with developing an understanding of the clients objectives must clearly understand his position. On the other hand, the client, who has ultimate accountability for results, often has little, if any, data processing knowledge or ability to deal effectively with the data processor.

Given this framework, how do data processor and client personnel communicate effectively? Can the data processing systems developed perform as required? Can specific performance measurement criteria be developed?

A plan must be developed to help both the data processor and client address these questions. The basic principles of the plan must be simple and capable of application across a wide range of client requirements.

STANDARDS OF PERFORMANCE TECHNIQUE

Given a corporation's major investment in data processing systems and resources, an effective process to ensure required human intervention can consist of the following steps:

- Establish "Standards of Performance" for each data processing system
- Measure the system's performance as it compares with each standard each month.
- Recognize and reward the performance contributions of both data processor and client personnel.

In labor-intensive corporations, effective expense control programs must concentrate on salary costs and, in turn, on the number of personnel required in a support role. This is the critical controllable expense.

I have found that where "Standards of Performance" are instituted, people are motivated to produce more than the results desired. The data processor and client begin to communicate effectively. And why not? For the first time they have a common base—a commonly understood and held set of measurement criteria.

ILLUSTRATIONS

A case in point: One of the larger data processing systems I have personally been associated with was an employee human resource data bank which supported both Personnel and Payroll functions of one of the largest multiple-line insurance companys in the United States. The system accommodates the personnel/payroll records of an active staff of over 30,000 employees throughout the United States, Canada, and Puerto Rico.

In the course of a year, the system manipulates, calculates and processes in excess of 1.2 million transactions. Associated with this is the annual production and distribution of over 350,000 personnel reports and 860,000 paychecks.

An integral part of the system was a client staff of 39 employees who were charged with the specific responsibility of inputting, pre-editing, coding, balancing, researching, analyzing, distributing and controlling this employee data. A staff of 19 data processors were required to support the technical aspects of the system. (Exhibit I illustrates the relationships.) The "system" in a real sense consists not only of the data processing equipment, but the manual support functions that people must perform.

Two years after installation, the system was perceived to be performing unfavorably. The data processors could not consistently satisfy the Payroll and Personnel client's "needs." Work was backlogged. Flexibility to handle new demands was limited. Processing costs were high—service levels, poor. Problems of this nature continued in spite of a 33 percent addition to staff over a two year period. The problem diagnosed was a lack of advanced data processing technology.

At this point, a Project Team, composed of data process-

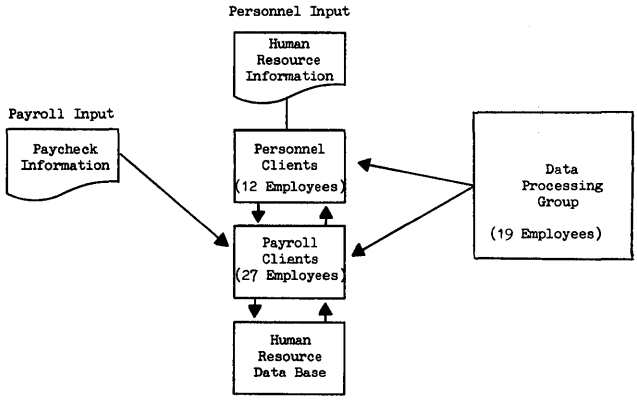


Exhibit I—Work relationships before standards of performance

ing and client personnel, was commissioned to investigate the feasibility of installing a more sophisticated technology. The first step taken by the team was to develop a set of objective criteria, against which system performance could be measured. These were termed "Standards of Performance." Using them as a basis, the existing technology was evaluated. To everyone's surprise, the existing system met the desired performance levels. In short, while the perception of poor performance existed, when evaluated against specific, desired criteria, we were able to demonstrate that the system was performing satisfactorily.

Unfortunately, as is often the case, it's the perceptions that people have that counts. Our real problem therefore, was one of education—to correct an erroneous judgment. The culprit was identified as a real lack of understanding of system requirements and capabilities from both the data processor and client perspectives. The data processor was expending a large proportion of effort on activities that were not critical to the client personnel's operations. In turn, the clients were not communicating their essential needs to the data processor. To further compound the problem, in the final analysis, the Human Resource System was in fact adequately servicing the end users and was more than meeting their expectations. In the course of the project team's interview process all end users contacted were in agreement on this point. In spite of a constantly increasing input volume the system continued to produce secure, controlled, timely, and accurate output.

The Project Team recommended that an "Operational Improvement" was warranted rather than installing a new data processing technology. That is, all manual input handling activities be combined into a single unit, with efficient workflow and procedures. Work measurement controls were installed to insure high productivity levels. In addition, the team recommended maintaining a strict adherence to the Human Resource System's "Standards of Performance."

The recommendations were accepted by management and vigorously implemented. Immediate cost avoidance stemming from cancellation of the projected data processing developmental effort was $200,000. More importantly, however, system performance was perceived to be and was in fact meeting specific expectations!

A longer term benefit was shown when two years after performance standards were implemented, a 50 percent reduction in data processor and a 10 percent reduction in client resources were achieved as a direct result of improved communications and working relationships. Clearly, the catalyst was an agreed upon set of performance standards. (Exhibit II illustrates the results.)

I don't pretend to suggest that all data processing problems are the result of erroneous perceptions or that they can be readily solved. Another case in point—performance standards were developed which, when applied, soon highlighted the fact that results were less than desired. An in depth probe revealed:

- Standards were ignored. While performance criteria were sound, they were ignored or inconsistently applied.
- Management placed little emphasis on monitoring actual results. This was visible to the employees since normal supervisor recordkeeping was delegated downward.
- Results were not reviewed between the data processor and client personnel. No ongoing constructive dialogue existed.

When the results of the review were presented to management they were accepted and constructively addressed. Within three months performance had returned to satisfactory levels. Again this was accomplished without the need to launch a new data processing system's development effort.

PERFORMANCE MEASUREMENT BENEFITS

When "Standards of Performance" are installed in an area, their impact while subtle, is dramatic in terms of results. For the first time the data processor and client have the same reliable and objective criteria on which to evaluate their collective performance. Usually some disturbing facts surface:

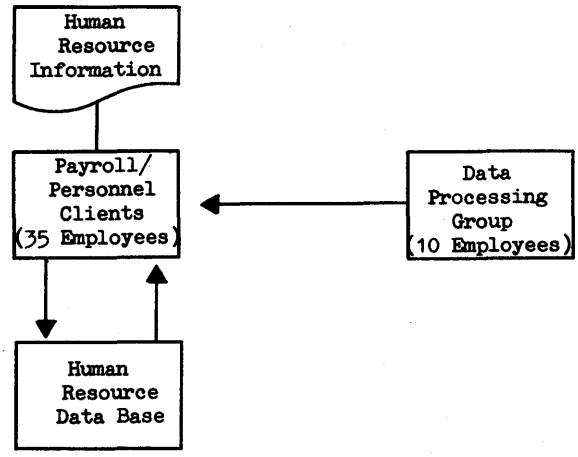


Exhibit II—Work relationships after standards of performance

- The data processing supervisor is not normally allocating his personnel to the critical client tasks. Too much attention is focused on mechanical efficiency.
- Advancements in technology are not usually an answer to performance problems.
- The client realizes he has not effectively utilized his available data processing resource. This realization can be a determining factor in greatly extending the life span of an existing data processing system, while at the same time realizing an attendant increase in performance. The cost avoidance, associated with not developing replacement systems is almost always substantial.

Once the standards are in place, communication between client and data processor and repetitive performance feedback is assured. This dialogue is an essential product of the "Standards of Performance" technique. It is the key to developing a balanced, factual perspective to data processing evaluation.

Positive results are visible in terms of reduced overtime and work backlogs. When standards are used, effective client/data processor communications are established. There is almost always a significant reduction in data processor resources required. Individual and group performance can be judged based on objective measurable results. Managers are in a better position to recognize and reward performance (or lack thereof). The net impact is a reduction in cost for salary expense, while, at the same time realizing an improvement in system performance.

STANDARDS DEVELOPMENT APPROACH

The approach employed to arrive at a data processing system's "Standards of Performance" is based on applying the management principles taught in the Louis A. Allen Project Seminar (Management Planning and Control) to the data processing environment.

"—ESTABLISH THE KEY OBJECTIVE: Identify the key data processing environment objective.
—ESTABLISH CRITICAL OBJECTIVES: Statements of the most important (continuing) results which must be accomplished if the key objective is to be realized.
—DEVELOP STANDARDS: Standards developed for each critical objective establish the criteria for effective performance."[1]

The crucial management ingredient underlying this approach is the need to reach understanding and agreement between data processor and client.

STANDARDS DEVELOPMENT ILLUSTRATION

"Standards of Performance" for a hypothetical career management system could be constructed as follows:

KEY OBJECTIVE

To provide a reporting framework which supports the development, installation, and maintenance of personnel career management programs within the corporation. Further, to be responsive to report users' status and information needs.

CRITICAL OBJECTIVE 1

To provide calculation and report development flexibility so that personnel career management reports can be tailored to report user needs on a timely, economical basis.

STANDARDS

- To develop new reports in not more than 34 elapsed days.
- To modify existing reports in not more than 22 elapsed days.
- To enable client report generation in not more than 10 elapsed days.
- To set-up a new employee "On File" in not more than three elapsed days.
- To revise input forms in not more than 11 elapsed days.

Report development performance measures are intended to eliminate the inefficient usage of reports. Emphasis is placed on delivering exception reports and summary level reports rapidly. Flexibility must be developed so that report users do not feel a need to request voluminous detailed reports. This need is normally generated in reaction to an unresponsive or a costly developmental process.

Specific measurement criteria can be visualized in terms of a two dimensional matrix. (Exhibit III illustrates sample relationships). Procedurally, this matrix can be developed in the following three steps:

STEP 1—*REPORT REQUIREMENTS* are analyzed in terms of critical client needs. Report categories must be clearly defined and capable of being measured. Additionally, the data processor and client must be in complete agreement with the categories selected.
STEP 2—*FUNCTIONS* that must be performed, to insure error-free reports, are documented for all participants. Typically this will include report recipients in addition to the data processor and client. It is usually found that lines of responsibility are not clearly defined, which causes inefficiencies and duplication of effort.
STEP 3—*ELAPSED TIMES* must be assigned for each REPORT REQUIREMENT and will reflect past performance and future requirements. Times must be reasonable, attainable and should reflect a common, understood, and agreed to goal.

The end result of setting report development measurement criteria will be a common understanding of the report development process for both the client and data processor, along with supporting a framework for client communications with report requestors. The client can now pursue, on

Function	REPORT REQUIREMENTS				
	Develop New Reports	Modify Existing Reports	Client Report Generation	New Employee Set-Up	Revision of Input Forms
Initial Report User Contact	1*	1	1	1	1
Finalize Specifications Report User & Client	1	1	1	2	10
Finalize Specifications Client & Data Processor	5	3	-	-	-
Analysis & Costing Client & Data Processor	5	3	-	-	-
Programming	10	5	2	-	-
Unit Test Data Generation (Client)	5	3	3	-	-
Unit Testing	3	2	2	-	-
Verify Results (Client)	1	1	1	-	-
Report Acceptance (Data Processor)	3	3	-	-	-
TOTALS	34	22	10	3	11

* All time estimates are in terms of elapsed days.

Exhibit III—Personnel career management report matrix

a factual basis, report delivery dates and can advise on the risks involved in attempting to unreasonably accelerate delivery dates.

CRITICAL OBJECTIVE 2

Complete data processing production runs accurately and on schedule.

STANDARDS

- 95 percent of all production runs will be completed within five (5) working days from receipt of the latest input item or input cut off date. This will apply unless other specific schedules are established.
- Machine resources will not exceed $5,000 monthly.

These measurement criteria are intended to be applied after the initial report development process is completed. The measurements ensure that recurring reports are delivered on a timely basis—it makes no sense to develop a report in a rapid and cost effective manner and then not monitor the recurring delivery process.

CRITICAL OBJECTIVE 3

Implement minor report changes (1-10 days) on a timely basis.

STANDARDS

- Resolve 98 percent of priority "A" changes prior to the next update (unless other arrangements agreeable to client and data processor have been made).
- All changes will be prioritized through the combined efforts of client and data processor personnel. Adding changes after the monthly schedule is established will result in a re-prioritization of existing planned work. Monthly planning of this activity should result in 95 percent of scheduled changes being completed without change.
- Implement 95 percent of all priority "B" (Balancing

problems, report format/content changes) changes and 90 percent of all priority "C" (noncritical) changes on a monthly basis as reflected on the agreed-upon monthly maintenance schedule.

It must be noted that for proper application of these standards the prioritization process must be agreed to and understood by both client and data processor. The priority process is negotiable, however typical priority definitions are as follows:

A. Essential to next production processing cycle.
B. Desirable change—but not essential to next cycle process.
C. Changes to improve system performance, control, input and report processing.

Once the reports are developed and the ongoing data processing system is installed, requirements for system change occur. The origin of change is complex and can be generated from report user, client, or data processor personnel. However generated, change requirements must be controlled in a manner that identifies the most critical requirements. The intent is to meet maintenance measurement criteria while minimizing client and data processor staffing needs.

DEVELOPMENT TIMING

Defining a system's standard of performance benefits both the client and the data processor. Ideally these performance criteria should be established early in a data processing system's developmental process. The standards ensure that the client fully communicates critical needs. Since the standards are normally given in terms of response times (elapsed days) to perform an activity, the data processor has the necessary information to answer difficult system design questions—while fulfilling customer requirements with a minimum investment of data processor resources.

The standards of performance can also be developed for systems already installed. Typically maintenance activities can generate an unacceptable level of resource commitment in response to undefined but assumed objectives. Again, the standards can document the clients critical needs and can serve as a vehicle to evaluate an existing system's performance. Closely monitored, it can also indicate a need to upgrade or replace the existing system.

INGREDIENTS FOR SUCCESS

The following four elements are necessary to ensure the success of the "Standards of Performance" technique.

STEP 1—ESTABLISH STANDARDS OF PERFORMANCE: The critical activity is to first establish the standards. Without sound standards the technique will fail. These performance standards cannot normally be established by methods such as time ladder studies, random sampling, stopwatch techniques, or modal time analysis. In other words the type of standards we are dealing with are not translatable to engineered time standards; reliance is placed on a combination of historical trends and consensus agreement.

STEP 2—PURSUE DATA PROCESSOR AND CLIENT UNDERSTANDING AND AGREEMENT: Having developed the standards, we must now set agreed to elapsed times and arrive at a complete understanding of requirements in both data processor and client areas. This must be done at the highest management levels in both areas. Again, without complete understanding and agreement, the chances of success are reduced.

STEP 3—REGULARLY MONITOR THE PERFORMANCE CRITERIA: The performance results must be discussed and acted upon on a monthly basis. This dialogue is yet another ingredient in ensuring the success of the technique since most standards cannot be as precise as an engineered time standard. A continuing reevaluation is the method used to test the standards as initially established and to answer such questions as: ARE RESULTS MEETING STANDARDS? IF NOT, WHY NOT?

STEP 4—REFINE STANDARDS WHEN NECESSARY: Given the imprecise nature of the standards, we must be willing to reevaluate them. Many external factors, such as client reorganizations or end user requirement changes can render some standards obsolete. Without continuous evaluation and updating all gains initially made can be lost.

CONCLUSION

No data processing effort should be attempted without first establishing "Standards of Performance." Differing client requirements and objectives may require a unique set of performance criteria, however, there is no doubt that measurement criteria are mandatory.

The challenge to improve is a shared responsibility. Here is a new technique which, if properly applied, will result in high "system productivity" and the achievement of clearly defined levels of performance with communications which effectively link all parties of "The System." Remember "The System" consists of automatic data processing equipment plus the personnel that make it go and *Perceptions*. While it is important to understand and control the people/machine relationship, it is equally as important that client and data processor share a common perception of performance.

REFERENCES

1. Allen, Louis A., *Management Planning and Control (AMPAC) Workbook*, 1976.
Myers, M. Scott, *Every Employee a Manager*, New York, McGraw-Hill Book Company, 1970.
Radamaker, T., ed., *Business Systems*, 2 Vols. Cleveland, Systems and Procedures Association, 1963.

Effects of peripheral processor wait list positioning on system performance

by RONNIE G. WARD, BECKY B. TURNER and GALEYN JOE HUBBARD
University of Texas at Arlington
Arlington, Texas

INTRODUCTION

Strauss[1] developed a simple analytic model of the HASP Execution Task Monitor (HETM) for use in studying performance improvements that result from HETM's application and whether or not HETM should be modified to consider different memory speeds of OS/MFT partitions at a specific installation. The cyclic-server multiprogramming model he developed is demonstrably valid.[2] That is, using service rates fitted to the model, predicted results approximate measured results reasonably well, and one may have confidence that the model accurately reflects system performance.

In IBM's OS, one or more job classes are assigned to each system initiator program, which selects jobs to introduce into the system on a priority basis within class. OS uses preemptive priority CPU task dispatching. Tasks are represented to the operating system using a task control block (TCB). The set of TCB's corresponding to active tasks are chained in priority order (highest to lowest). The dispatcher assigns the CPU to the first ready task found by scanning the TCB chain in priority order. An event completion that causes a higher priority task to become ready generates a task switch and a running lower priority task will be preempted. HETM regularly examines the CPU utilization of OS tasks and rearranges the TCB chain in an attempt to more equitably distribute CPU service. As Strauss points out, the effect is to keep I/O bound tasks active without unduly harming service to CPU bound tasks.

When an executing program solicits an I/O operation, the I/O supervisor[3] associates a request element (RE) with the intended work. If the I/O operation cannot be started immediately, the RE is positioned in a wait list associated with the addressed device. An installation may specify at system generation time the wait list positioning policy to be used for each device. For example, an installation may choose for DASD devices either first-come first-served (FCFS) queuing, dispatching priority positioning, or seek address positioning.

The purpose of this paper is to quantify in terms of the results of an analytic model the effect wait list positioning can have on system performance. In systems where CPU or channel time is a bottleneck, the model can be used to study the importance of wait list positioning in increasing or decreasing job class response. The analytic model of different wait list positioning strategies is first presented. The HETM stability condition given in Reference 1 is then re-examined. Finally, performance predictions of the positioning policies are discussed and related to system and job class performance.

THE ANALYTIC MODEL

Figure 1 depicts a simple cyclic exponential server model that is a closed network permitting no exogenous arrivals or departures. There are three task classes α, β, and γ specified by their CPU service rates μ_i and peripheral service rates δ_i where i is α, β, or γ. The characters a, b, and c denote the tasks ranked by preemptive CPU priority; task a can preempt b which can preempt c for CPU service. A mapping is defined as a permutation of the α, β, γ tasks onto the priority designations a, b, c. It is the HETM reordering of the OS TCB chain that determines a particular mapping. The tasks a, b, and c are serviced by an exponential process CPU server with mean rate μ_i. Following CPU service, the task enters a peripheral processor (PP) wait list and eventually receives exponential process peripheral service with mean rate δ_i. After PP service, the task may enter the CPU queue or be serviced by the CPU. The tasks cycle continuously through a CPU-PP service loop.

The system is modeled in steady state during HASP intervals between changes to the TCB chain. Analytic and simulation studies have shown this to be a reasonable assumption.[1]

In the model, each of the tasks can be in any one of the following five places in the system:

0. Waiting for CPU service.
1. Obtaining CPU service.
2. At the end of the PP wait list.
3. At the head of the PP wait list.
4. Obtaining PP service.

The state of the system is completely specified by the contents of four of the five positions. A four component state

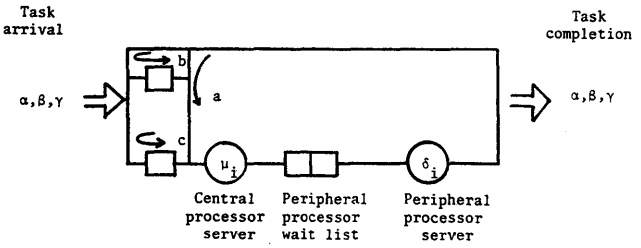


Figure 1—Cyclic exponential server model

vector v is used to denote the system state where the jth component of v denotes the jth position of the system and the contents of each component can be a, b, c, or x (x denotes an empty position). For example, the state $v=axbc$ indicates that a "class a" task is receiving cpu service, a "class b" task is at the head of the I/O queue, a "class c" task is receiving PP service, and the second position in the PP wait list is unoccupied.

Using this notation, the possible state vectors v_i^k of five PP wait list positioning strategies are written as follows:

$$v_{1\le i\le 16}^{1} = \begin{array}{c} 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16 \\ \hline \left\{\begin{array}{l} a\ b\ c\ a\ x\ c\ a\ a\ x\ b\ x\ b\ x\ a\ x\ x \\ x\ x\ x\ x\ c\ x\ x\ x\ c\ x\ a\ x\ b\ x\ b\ a \\ x\ x\ b\ x\ b\ a\ c\ x\ a\ c\ c\ a\ c\ b\ a\ b \\ x\ a\ a\ b\ a\ b\ b\ c\ b\ a\ b\ c\ a\ c\ c\ c \end{array}\right. \end{array}$$

FCFS and RANDOM positioning (k=1)

$$v_{1\le i\le 11}^{2} = \left\{\begin{array}{l} a\ b\ c\ a\ x\ c\ a\ x\ x\ b\ x \\ x\ x\ x\ x\ b\ x\ x\ a\ b\ x\ a \\ x\ x\ b\ x\ c\ a\ b\ c\ a\ a\ b \\ x\ a\ a\ b\ a\ b\ c\ b\ c\ c\ c \end{array}\right.$$

Last-come First-served (LCFS) (k=2)

$$v_{1\le i\le 12}^{3} = \begin{array}{c} 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16 \\ \hline \left\{\begin{array}{l} a\ b\ c\ x\ a\ a\ c\ x\ a\ b\ b\ x \\ x\ x\ x\ c\ x\ x\ x\ c\ x\ x\ x\ b \\ x\ x\ b\ b\ x\ c\ a\ a\ x\ c\ a\ a \\ x\ a\ a\ a\ b\ b\ b\ b\ c\ a\ c\ c \end{array}\right. \end{array}$$

Dispatching Priority Positioning (k=3)

$$v_{1\le i\le 10}^{4} = \left\{\begin{array}{l} a\ b\ c\ x\ a\ a\ c\ x\ x\ b \\ x\ x\ x\ b\ x\ x\ x\ a\ a\ x \\ x\ x\ b\ c\ x\ b\ a\ b\ c\ a \\ x\ a\ a\ a\ b\ c\ b\ c\ b\ c \end{array}\right.$$

Inverse Dispatching Priority Positioning (k=4)

The application of a wait list positioning policy other than FCFS is quickly observed by studying the state transitions. For example, $v_6^3 \rightarrow v_8^3$ since the a task would position ahead of the c task after completing CPU service. On the other hand, $v_7^4 \rightarrow v_9^4$ since an inverse priority scheme is used.

As pointed out in Reference 4, FCFS scheduling is an appropriate model of secondary storage I/O devices because preemptive scheduling is usually not possible or efficient. However, given that the device is not preemptable, task positioning in the device wait list can still influence the performance measures for an individual job class. Typically, analytic models presented in papers such as in References 5 and 6 assume that device scheduling is independent of the processes generating the requests. However, this may not always be the case as already discussed in the instance of OS.[3] With the above models, it is possible to determine analytically the impact of wait list positioning on job class performance. Note that LCFS scheduling is feasible only for interjob requests.

Each state v_i^k has an associated steady state probability $P_{v_i^k}$. These equilibrium probabilities can be determined by solving the system balance equations [1,4]. That is, for all states v_i^k,

$$\sum_j P_{v_j^k} \times (\text{rate of flow from } v_j^k \text{ to } v_i^k) = P_{v_i^k} \times (\text{rate of flow out of } v_i^k)$$

From these probabilities, the task CPU utilization, PP utilization, queue time, and cycling rates may be determined.[1,2] Also note that the distinction between FCFS and random positioning is in the state transitions and balance equations. For example, under FCFS, the c task exiting the CPU results in $v_3^1 \rightarrow v_5^1$. However, under random positioning we could also have the transition $v_3^1 \rightarrow v_{13}^1$ with an equal probability (.5). The flow into certain states is therefore altered. For instance, the balance equation for v_5^1 under FCFS is

$$\delta_a P_{xcba} = \mu_c P_{cxba}$$

and under random positioning we would have

$$\delta_a P_{xcba} = .5\mu_c P_{cxba} + .5\mu_b P_{bxca}$$

Another interesting model result aside from service center utilizations and task throughputs is the task switching rate TSW^k. A task switch occurs when a higher priority task becomes ready to use the CPU. For the above models, we have for TSW^1 and TSW^3:

$$\delta_a(P_{bxxa} + P_{cxba} + P_{bxca}) + \delta_b P_{cxab}$$

and for TSW^2 and TSW^4:

$$\delta_a(P_{bxxa} + P_{cxba}) + \delta_b P_{cxab}$$

Note that although the method of calculation may be the same, the results will differ because of different valued state probabilities. The TSW^k give an indication of the operating system overhead due to task switching. Naturally, the lower the value the lower the overhead. Similar overhead measures can be obtained for wait list positioning using a priority scheme.

HETM STABILITY CONDITION

At each HASP interval, HETM rearranges the subset of the OS dispatching chain constituted by an installation spec-

ified dynamic priority group. Tasks are ranked in inverse order of their CPU utilization histories. Equation (1) is used to determine the CPU utilization history $h_{t,n}$ of the nth task at HASP interval t.

$$h_{t,n}=\text{cpu}_{t,n}+h_{t-1,n}-H_t/N \tag{1}$$

N is the number of tasks in the dynamic priority group, $\text{cpu}_{t,n}$ is the CPU counts observed for the nth task during the tth HASP interval, and $H_t=\sum_{i=1}^{N}(\text{cpu}_{t,i}+h_{t-1,i})$.

Adopting the notational conventions of Reference 2:

$$\underline{h}_t=\begin{bmatrix} h_{t_d} \\ h_{t,\beta} \\ h_{t,\gamma} \end{bmatrix}, \quad C_t=\begin{bmatrix} \text{cpu}_{t,\alpha} \\ \text{cpu}_{t,\beta} \\ \text{cpu}_{t,\gamma} \end{bmatrix} \tag{2}$$

equation (1) can be rewritten for the three job model as:

$$\underline{h}_t=A(\underline{h}_{t-1}+C_t) \tag{3}$$

where

$$A=\frac{1}{3}\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

Strauss[1] gives the solution to (3) as:

$$\underline{h}_t=A\left(\sum_{j=1}^{t} C_j+h_0\right) \tag{4}$$

For a simple stable mapping (i.e. the mapping for the next HASP interval is the same as the previous) with $\underline{h}_0=0$, equation (4) reduces to

$$\underline{h}_t=tAC \tag{5}$$

where C is the CPU utilization vector associated with the repeating mapping.

For higher order repeating mapping patterns, Strauss[1] determines that if h_t is to repeat every $R>1$ control intervals, then h_{mR} must equal h_{nR} for all integers m and n. Thus from (4):

$$\underline{h}_{mR}=A\left(\sum_{j=(m-1)R+1}^{mR} C_j\right)+\underline{h}_{(m-1)R} \tag{6}$$

Clearly, for h_{mR} to equal $h_{(m-1)R}$ (and thus repeat the mapping pattern), we must have

$$A\left(\sum_{j=(m-1)R+1}^{mR} C_j\right)=0.$$

Since A is idempotent, the vector C^R of (7) must have equal components.

$$C^R=\begin{bmatrix} \text{cpu}_\alpha{}^R \\ \text{cpu}_\beta{}^R \\ \text{cpu}_\gamma{}^R \end{bmatrix}=\sum_{j=1}^{R} C_j \tag{7}$$

To relax the requirement that C^R have equal components consider the following. Let s_1, s_2, s_3 and s_1', s_2', s_3' be mutations of α, β, γ such that the mappings generated by $\underline{h}_t$ and $\underline{h}_{t'}$ $(t\neq t')$ imply $h_{t,s_1}\leq h_{t,s_2}\leq h_{t,s_3}$ and $h_{t',s_1'}\leq h_{t',s_2'}\leq h_{t',s_3'}$. Define $\underline{h}_t\sim\underline{h}_{t'}$ if and only if $s_1=s_1'$, $s_2=s_2'$, and $s_3=s_3'$. Then $\sim$ is an equivalence relation.

Let E be the equivalence class formed by $\underline{h}_{(m-1)R}$ (i.e. if $W=\{\underline{h}_t \mid t>0\}$, then $E=\{\underline{h}_t\epsilon W \mid \underline{h}_{(m-1)R}\sim\underline{h}_t\}$). Deriving a stability condition requires that $\underline{h}_{mR}\sim\underline{h}_{(m-1)R}$. In order to have $\underline{h}_{mR}\epsilon E$, we must have some permutation s_1, s_2, s_3 of α, β, γ such that

$$h_{mR,s_1}\leq h_{mR,s_2}\leq h_{mR,s_3} \tag{8}$$

and

$$h_{(m-1)R,s_1}\leq h_{(m-1)R,s_2}\leq h_{(m-1)R,s_3} \tag{9}$$

Thus, if we let k_1, k_2, k_3 be the permutation of 1, 2, 3 for the mapping s_1, s_2, s_3 corresponding to the α, β, γ index of h_t, from (8) we get:

$$\begin{aligned} A_{k_1}{}^*C^R+h_{(m-1)R,s_1}\leq A_{k_2}{}^*C^R \\ +\underline{h}_{(m-1)R,s_2}\leq A_{k_3}{}^*C^R+\underline{h}_{(m-1)R,s_3} \end{aligned} \tag{10}$$

where $A_i{}^*$ denotes the ith row vector of A. Using (9), we may guarantee (10) if

$$A_{k_1}{}^*C^R\leq A_{k_2}{}^*C^R\leq A_{k_3}{}^*C^R.$$

Adding $1/3(\text{cpu}_\alpha{}^R+\text{cpu}_\beta{}^R+\text{cpu}_\gamma{}^R)$ to each term yields

$$\text{cpu}_{s_1}{}^R\leq\text{cpu}_{s_2}{}^R\leq\text{cpu}_{s_3}{}^R.$$

We assume in the above discussion that equal history values imply that α precedes β which precedes γ.

PERFORMANCE COMPARISONS

This section discusses the relative performances of the five wait list positioning policies. The models presented in an earlier section were solved for the two sets of μ_i and δ_i presented in Table I.[1,2] These parameters are both interesting and representative. The μ_i and δ_i of Example 1 were derived by fitting the model to an OS/MFT system operating under HASP. Example 2 parameters are presented in Reference 1 and it can be seen that α is a light CPU, heavy I/O task; β is a moderate CPU, light I/O task; and γ is a heavy CPU, moderate I/O task. It should be noted that since the balance equations are homogeneous, a given set of utilizations can result from an infinite set of service rates. The relative values of the service rates are therefore important, and representative values such as those in Example 2 may be used.

The model predictions for all possible mappings using the parameters of Table I are presented in Tables II-VI for Example 1, and Tables VII-XI for Example 2. Using the derived stability condition, the different wait list positioning models were examined for stable mapping patterns through simple simulation. For Example 1, #4 appears to be a first order stable mapping when limiting on the history values[1] is imposed. For Example 2, #1 is a first order stable mapping. The model predictions are compared using these two stable mappings.

Clearly, with both sets of data and under all positioning policies, total PP utilization does not significantly vary. This results from the PP device not being preemptable. However, total CPU utilization fluctuates significantly, most noticea-

TABLE I.—Model Parameters

Task	Example 1 CPU Service μ_i (sec^{-1})	Example 1 PP Service δ_i (sec^{-1})	Example 2 CPU Service μ_i (sec^{-1})	Example 2 PP Service δ_i (sec^{-1})
α	6	11.94	25	5
β	99.94	18.43	15	75
γ	81	24.48	5	25

TABLE II.—Priority CPU Scheduling with FCFS PP Scheduling

Mapping	CPU Utilization				PP Utilization					Cycling Rates			
No. abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1. αβγ	.605	.033	.036	.674	.304	.181	.120	.605	1.279	.12	.60	.56	.49
2. αγβ	.606	.041	.029	.676	.304	.134	.160	.599	1.275	.12	.61	.55	.49
3. βαγ	.104	.490	.035	.629	.566	.246	.115	.927	1.556	1.35	1.74	.49	.47
4. βγα	.086	.090	.422	.599	.468	.298	.212	.978	1.577	1.67	1.44	1.22	.42
5. γβα	.109	.074	.419	.602	.362	.400	.211	.972	1.574	1.74	1.48	1.23	.42
6. γαβ	.150	.482	.028	.659	.496	.242	.150	.888	1.548	1.65	2.03	.48	.46

TABLE III.—Priority CPU Scheduling with Priority PP Scheduling

Mapping	CPU Utilization				PP Utilization					Cycling Rates			
No. abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1. αβγ	.623	.034	.029	.686	.313	.183	.097	.594	1.280	.12	.62	.56	.40
2. αγβ	.625	.041	.025	.692	.314	.136	.137	.587	1.279	.12	.63	.56	.42
3. βαγ	.110	.494	.028	.631	.596	.248	.091	.935	1.565	1.42	1.83	.49	.37
4. βγα	.095	.099	.279	.472	.517	.327	.140	.984	1.456	1.20	1.59	1.33	.28
5. γβα	.121	.081	.277	.478	.399	.440	.139	.979	1.457	1.28	1.63	1.35	.28
6. γαβ	.159	.485	.023	.667	.526	.244	.126	.895	1.562	1.75	2.15	.48	.39

TABLE IV.—Priority CPU Scheduling with Inverse Priority PP Scheduling

Mapping	CPU Utilization				PP Utilization					Cycling Rates			
No. abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1. αβγ	.488	.045	.057	.589	.245	.243	.188	.676	1.265	.20	.49	.75	.77
2. αγβ	.490	.054	.045	.589	.246	.179	.245	.670	1.258	.20	.49	.73	.75
3. βαγ	.095	.468	.053	.616	.517	.235	.175	.927	1.543	1.33	1.59	.47	.71
4. βγα	.079	.093	.478	.650	.427	.308	.240	.976	1.625	1.93	1.31	1.26	.48
5. γβα	.095	.076	.477	.649	.314	.416	.240	.970	1.618	1.92	1.28	1.28	.48
6. γαβ	.134	.452	.040	.627	.444	.227	.220	.891	1.518	1.57	1.81	.45	.68

TABLE V.—Priority CPU Scheduling with Random PP Scheduling

Mapping		CPU Utilization				PP Utilization					Cycling Rates			
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.610	.032	.036	.678	.306	.176	.119	.602	1.280	.13	.61	.54	.49
2.	αγβ	.610	.039	.030	.679	.306	.130	.161	.597	1.276	.13	.61	.53	.49
3.	βαγ	.104	.454	.043	.600	.565	.228	.142	.934	1.535	1.31	1.73	.45	.58
4.	βγα	.088	.087	.420	.595	.481	.286	.211	.978	1.573	1.67	1.48	1.17	.42
5.	γβα	.112	.072	.420	.603	.371	.390	.211	.972	1.575	1.74	1.51	1.20	.42
6.	γαβ	.148	.445	.034	.627	.489	.224	.185	.898	1.525	1.60	1.99	.45	.57

TABLE VI.—Priority CPU Scheduling with LCFS PP Scheduling

Mapping		CPU Utilization				PP Utilization					Cycling Rates			
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.549	.036	.050	.635	.276	.198	.165	.639	1.274	.17	.55	.61	.68
2.	αγβ	.565	.040	.038	.644	.284	.133	.207	.624	1.267	.16	.57	.54	.63
3.	βαγ	.101	.335	.069	.505	.549	.168	.229	.946	1.451	1.06	1.69	.33	.93
4.	βγα	.085	.082	.479	.646	.465	.271	.240	.976	1.622	1.87	1.43	1.10	.48
5.	γβα	.104	.071	.477	.652	.343	.386	.240	.969	1.621	1.93	1.40	1.19	.48
6.	γαβ	.139	.334	.053	.526	.458	.168	.290	.916	1.442	1.28	1.87	.33	.89

TABLE VII.—Priority CPU Scheduling with FCFS PP Scheduling

Mapping		CPU Utilization				PP Utilization					Cycling Rates			
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.159	.204	.309	.672	.793	.041	.062	.895	1.567	2.79	3.96	3.06	1.55
2.	αγβ	.158	.400	.104	.663	.789	.080	.021	.890	1.553	2.60	3.95	2.00	1.57
3.	βαγ	.546	.087	.180	.813	.109	.434	.036	.579	1.392	8.65	8.19	2.17	.90
4.	βγα	.778	.171	.013	.961	.156	.034	.067	.256	1.218	11.57	11.66	.85	.33
5.	γβα	.759	.109	.033	.901	.152	.022	.165	.338	1.239	3.02	3.80	1.63	.82
6.	γαβ	.683	.070	.068	.821	.137	.349	.014	.500	1.320	2.77	3.41	1.75	1.02

TABLE VIII.—Priority CPU Scheduling with Priority PP Scheduling

Mapping		CPU Utilization				PP Utilization					Cycling Rates			
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.160	.205	.278	.643	.802	.041	.056	.898	1.541	2.63	4.01	3.07	1.39
2.	αγβ	.159	.400	.083	.642	.797	.080	.017	.893	1.536	2.49	3.98	2.00	1.24
3.	βαγ	.551	.087	.161	.799	.110	.436	.032	.578	1.377	8.59	8.26	2.18	.80
4.	βγα	.782	.170	.013	.965	.156	.034	.065	.256	1.220	11.73	11.73	.85	.33
5.	γβα	.765	.108	.032	.905	.153	.022	.162	.337	1.242	3.82	3.82	1.62	.81
6.	γαβ	.689	.069	.055	.813	.138	.346	.011	.494	1.307	2.44	3.44	1.73	.82

TABLE IX.—Priority CPU Scheduling with Inverse Priority PP Scheduling

Mapping		CPU Utilization				PP Utilization						Cycling Rates		
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.158	.200	.313	.671	.791	.040	.063	.894	1.565	3.72	3.96	3.00	1.57
2.	αγβ	.157	.398	.128	.683	.785	.080	.026	.890	1.572	3.20	3.92	1.99	1.92
3.	βαγ	.491	.092	.209	.792	.098	.461	.042	.601	1.394	8.18	7.37	2.31	1.04
4.	βγα	.757	.165	.016	.938	.151	.033	.082	.266	1.205	11.41	11.35	.83	.41
5.	γβα	.748	.100	.033	.881	.150	.020	.166	.335	1.216	3.42	3.74	1.50	.83
6.	γαβ	.653	.073	.092	.818	.131	.365	.018	.514	1.332	2.91	3.26	1.83	1.38

TABLE X.—Priority CPU Scheduling with Random PP Scheduling

Mapping		CPU Utilization				PP Utilization						Cycling Rates		
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.160	.161	.327	.648	.799	.032	.065	.896	1.544	2.97	3.99	2.41	1.64
2.	αγβ	.159	.360	.132	.652	.795	.072	.026	.893	1.545	2.90	3.97	1.80	1.99
3.	βαγ	.494	.092	.208	.794	.099	.461	.042	.601	1.395	8.22	7.41	2.30	1.04
4.	βγα	.765	.159	.016	.940	.153	.032	.079	.264	1.203	11.43	11.47	.80	.39
5.	γβα	.758	.085	.035	.878	.152	.017	.173	.342	1.220	3.02	3.79	1.28	.87
6.	γαβ	.658	.073	.092	.822	.132	.363	.018	.512	1.334	2.92	3.29	1.81	1.37

TABLE XI.—Priority CPU Scheduling with LCFS PP Scheduling

Mapping		CPU Utilization				PP Utilization						Cycling Rates		
No.	abc	a	b	c	Σ cpu	a	b	c	Σ pp	System Total	TSW	a	b	c
1.	αβγ	.159	.193	.303	.655	.797	.039	.061	.896	1.551	3.05	3.98	2.90	1.51
2.	αγβ	.159	.383	.108	.650	.794	.077	.022	.892	1.542	2.72	3.97	1.92	1.61
3.	βαγ	.531	.089	.186	.805	.106	.445	.037	.588	1.393	8.47	7.96	2.22	.93
4.	βγα	.771	.166	.015	.951	.154	.033	.073	.261	1.212	11.52	11.57	.83	.37
5.	γβα	.759	.102	.033	.894	.152	.020	.164	.336	1.230	3.19	3.80	1.53	.82
6.	γαβ	.687	.069	.072	.829	.137	.347	.014	.499	1.328	2.77	3.44	1.74	1.08

bly in Example 1. For this case, the *c* task spends more time in the wait list under priority positioning and total system utilization is lowest, but throughput is maximized. Situated there, the *c* task cannot get to the CPU, thereby lowering total CPU utilization. Likewise, since both the *a* and *b* tasks position ahead of it in the wait list and given that their CPU use is comparatively light, their throughput jumps, raising overall system throughput. Obviously, *a* and *b* task turnaround time decreases while the *c* task's increases. Furthermore, task switching overhead drops under priority positioning. Again, this occurs since the *c* task is in the PP wait list a larger percentage of the time.

Over the range of μ_i to δ_i, with equal job class characteristics, the *a* task will have better CPU and PP utilization under priority wait list positioning. This results in a lower response time and increased throughput. The same holds true for the *c* task under inverse priority positioning. Under priority positioning, the *c* task will always suffer greater response time and less throughput as does the *a* task under inverse positioning. This can be observed in Figures 2 and 3.

These results suggest a desirable scheduling strategy for either CPU bound systems, or more likely, batch systems supporting an interactive computing load. Faster turnaround will result for the high priority interactive requests at the expense of lower priority (batch) jobs. This is especially true

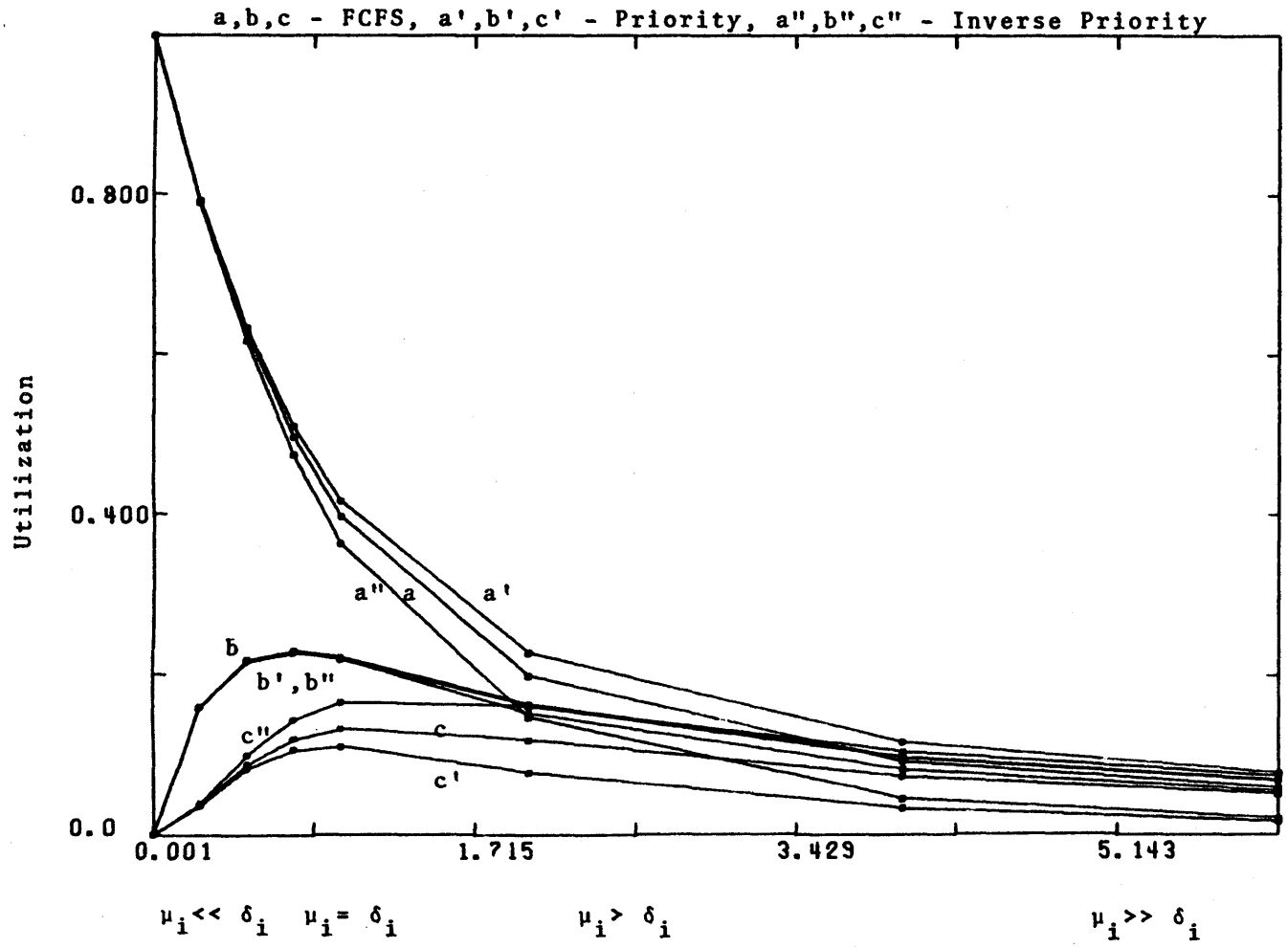


Figure 2—CPU utilization with equal job class characteristics over the range of service rates

for OS since it was not initially designed to support conversational systems.

When priority wait list positioning is utilized under OS with HASP, a curiosity develops. At the time a task is created, OS assigns it a dispatching priority. At the University of Texas Regional Computer Center all batch job tasks are created with the same dispatching priority. Since HETM does not alter the dispatching priority when it rearranges the TCB chain and the I/O Supervisor makes use of it to determine queue position, then, in effect, random I/O scheduling occurs. Moreover, even if different priority values were assigned (or changed through the CHAP macro), I/O scheduling would not always reflect HETM's current arrangement of the TCB chain. In essence, this is a HETM shortcoming. The TCB chain ordering represents the most recent information regarding task characteristics, which are known to change dramatically during execution. By not updating dispatching priorities to coincide with the TCB chain ordering, under priority wait list positioning HETM is ignoring an aspect of the problem it was intended to solve.

With inverse dispatching priority positioning, it can be observed that neither *a* nor *b* task throughput is adversely affected. As one would expect, CPU utilization increases with this positioning policy. This results because the CPU bound job (α) is assigned to the *c* task. Given that PP use remains relatively unchanged, total system utilization will tend to maximize under this policy. This can be seen in the two examples. Also, observe that a more balanced cycling rate is a consequence. Inasmuch as the outcomes of this policy are not calamitous for the *a* task, systems with an under utilized CPU may find this scheduling strategy attractive. The model also predicts, however, an increase in operating system overhead due to task switching.

Though inferences concerning real system performance are being made, it would be inappropriate to cite percentage changes in the models' predicted utilizations. This would attach an unsupported significance to the model results. This kind of observation can only be made in a specific situation using a perhaps more detailed model (one with additional PP devices and more job classes) with fitted service rates. Four job models were formulated and solved using representative service rates and the above observations remain true with

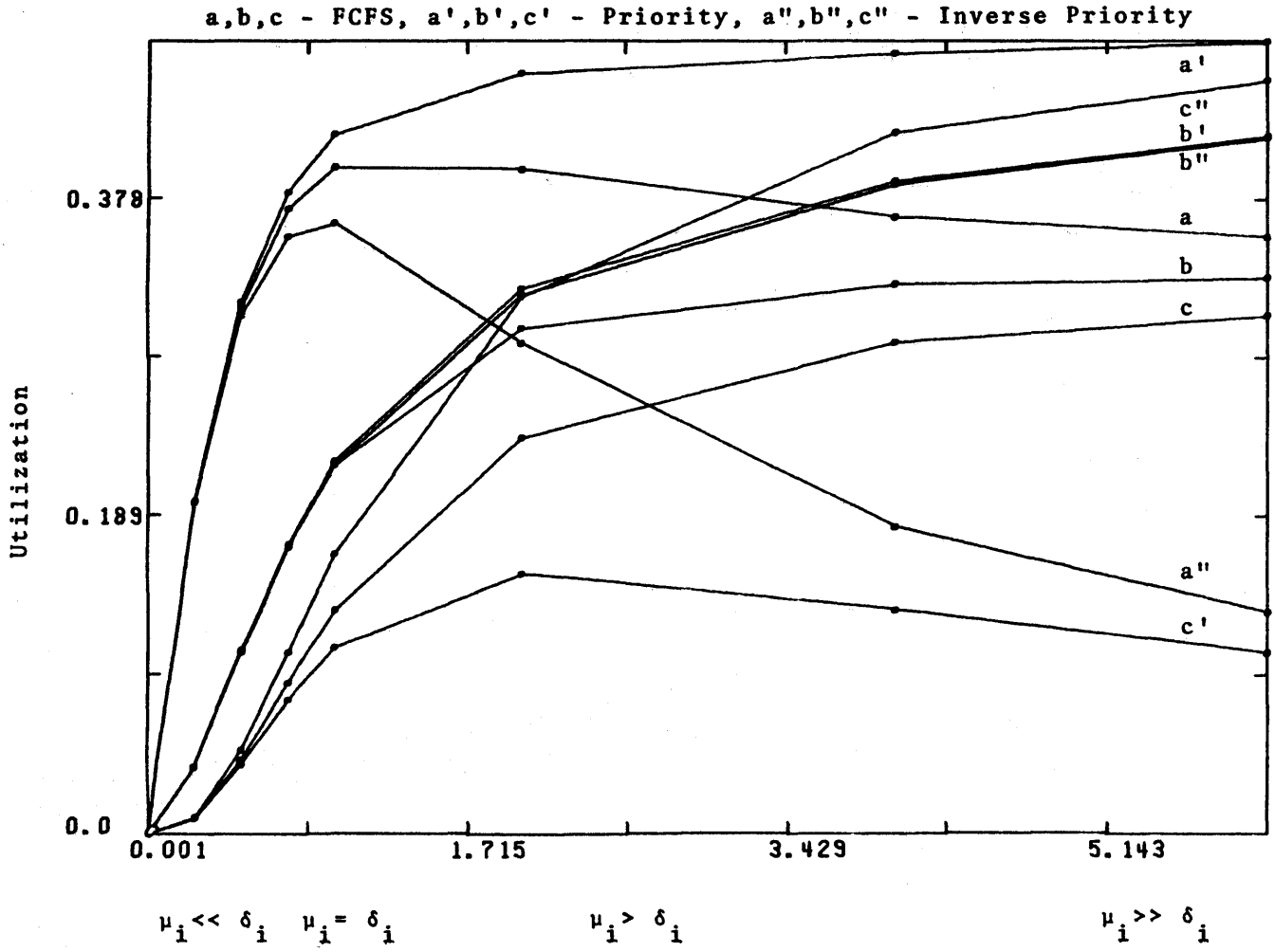


Figure 3—PP utilization with equal job class characteristics over the range of service rates

marked changes in the percentage differences of the predicted utilizations. The details of these models are not presented.

CONCLUSIONS

Five peripheral processor wait list positioning policies are presented. Relative performances are compared using a simple analytical model of a multiprogramming system with preemptive priority CPU dispatching on different classes of jobs.

Using fitted[2] service rates, priority wait list positioning was found to maximize throughput while minimizing total system utilization. High priority jobs receive better throughput and decreased response time. On the other hand, inverse dispatching priority positioning tends to maximize total system utilization and is not ruinous to high priority job response.

ACKNOWLEDGMENT

The authors are grateful for assistance provided by Jon C. Strauss.

REFERENCES

1. Strauss, J. C., "An Analytic Model of the HASP Execution Task Monitor," *Communications of the ACM*, Vol. 17, No. 12, December 1974, pp. 679-685.
2. Wong, K. and J. C. Strauss, "Use of a Software Monitor in the Validation of an Analytic Computer System Model," *Software-Practice and Experience*, Vol. 4, 1974, pp. 225-263.
3. OS I/O Supervisor Logic, IBM Corporation, GY 28-6616.
4. Baskett, F., K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the ACM*, Vol. 22, No. 2, April 1975, pp. 248-260.
5. Denning, P. J., "Effects of Scheduling on File Memory Operations," *AFIPS Conference Proceedings*, Vol. 30, 1967, pp. 9-21.
6. Teorey, T. J., "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," *AFIPS Conference Proceedings*, Vol. 41, 1972, pp. 1-11.

BEST/1™—Design of a tool for computer system capacity planning

by J. P. BUZEN, R. P. GOLDBERG, A. M. LANGER, E. LENTZ,* H. S. SCHWENK, D. A. SHEETZ and A. SHUM**

BGS Systems, Inc.
Lincoln, Massachusetts

INTRODUCTION

This paper presents the design philosophy used in the BEST/1 computer performance evaluator. Essentially, BEST/1 is a modeling tool which has been developed to address the "what if" questions that arise when evaluating computer system performance. These questions are of importance in such areas as capacity planning, performance tuning, hardware vendor selection and new system design.

From a design standpoint, the most significant features of BEST/1 are those which relate to ease of use. The following five objectives were considered most essential in this regard:

- The analyst who is familiar with the basic terms and concepts of computer performance evaluation should require no special training in order to use BEST/1. That is, the internal algorithms and procedures used by BEST/1 should be insulated from the user to the maximum extent possible.
- The level of effort required to set up and evaluate a BEST/1 model should be commensurate with the analyst's objectives. First order models should be easy to set up and evaluate, with more refined models requiring progressively more effort in terms of data collection and analysis.
- The input data required by BEST/1 should be easy to understand and obtain. Detailed measurements of internal system activity should not be necessary when studying simple questions.
- BEST/1 should operate interactively so that an analyst can have the benefit of working through a problem and exploring a number of related "what if" questions during a single analysis session. The speed requirements associated with interactive operation are also important in sensitivity studies where a wide range of parameter settings have to be explored.
- BEST/1 should generate a set of output reports which are easy to understand and apply.

In order to achieve these objectives, the BEST/1 design is based on an approach which is commonly employed in applications-oriented higher level languages. That is, a user visible front end has been developed which enables analysts to specify models and carry out analyses using concepts and terms which are generally familiar to individuals working in the area of computer performance evaluation. This user visible front end is then automatically translated into an internal BEST/1 analysis model which is insulated from the user in much the same way as machine language programs are insulated from the user of any higher level language. Thus, the most important design features of BEST/1 are those which relate to its user visible front end. These features are discussed in detail in the remainder of this paper.

It should be noted that, in order to meet the objective of interactive operation, the internal BEST/1 analysis models are based on extensions to the theory of multiple class queueing network models. However, the BEST/1 user need not be concerned with the mathematical details of this theory, just as the user of an applications-oriented higher level language need not be concerned with the details of a particular machine language. For this reason, the mathematical foundations of BEST/1 will not be emphasized in this paper. For discussions of the theory, see the references listed in the bibliography.

BEST/1 MODELING CONCEPTS

BEST/1 is an analysis tool which is capable of calculating the performance of almost any computer system on the basis of parameters which characterize that system's hardware, software, and workload. The performance factors which are calculated by BEST/1 include throughput, response time, queue length, processor utilization, and so on. See Figure 1.

The performance calculation function performed by BEST/1 is similar to the function performed by a benchmark experiment. That is, a BEST/1 analysis and a benchmark experiment both enable the analyst to evaluate the way a computer system will perform under a specific set of conditions. However, in the case of a benchmark experiment

* Currently affiliated with Booz, Allen Hamilton, Inc., Los Angeles, California
** Currently affiliated with IBM Corporation, San Jose, California

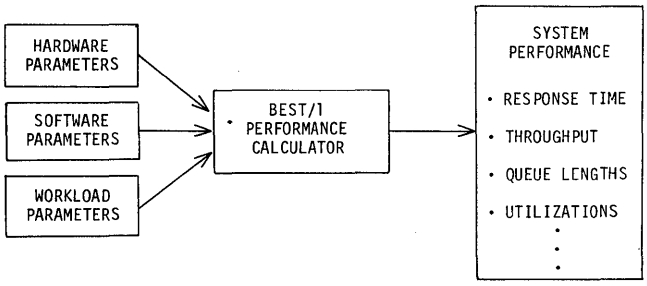


Figure 1—Basic BEST/1 functionality

this evaluation is made by running a set of programs on an actual system. In contrast, BEST/1 makes this evaluation on the basis of certain numerical parameters which are either measured directly or estimated by the analyst.

THE WORKLOAD CONCEPT

As illustrated in Figure 2, BEST/1 is based on the concept that performance issues arise when workloads of various types are processed by a computer system. In this context, the term "workload" refers to a stream of jobs or transactions arriving at a computer system from some external source. For example, a stream of batch jobs submitted by local or remote users could constitute a batch workload, a stream of transactions generated by time sharing terminals could constitute a time sharing workload, and so on.

The significance of the workload concept lies in the fact that analysts are usually most concerned with performance factors which are defined on a "per workload" basis. This is because users normally perceive throughput and response time as being associated with individual workloads rather than with the system as a whole. In fact, it is generally meaningless to discuss throughput or response for an entire system in cases where several workloads are present.

BEST/1 explicitly recognizes the central importance of workloads and uses the workload concept as a focal point for developing system models and organizing input parameters. In particular, BEST/1 allows the analyst to specify a model in terms of certain generic workload types. Each workload type has an associated set of parameters which characterize the special features of that workload. The three principal workload types recognized by BEST/1 are discussed below.

TIME SHARING WORKLOADS

The general nature of interactive or time sharing workloads is illustrated in Figure 3. Essentially, each time sharing workload is associated with a set of terminals. The users at the terminals are assumed to generate transactions which arrive at the system and undergo processing. When a transaction has been completed, a response is printed on the user's terminal. The user then generates a new transaction and the cycle repeats.

Figure 2—The workload concept

In the case of time sharing workloads, the analyst specifies the number of terminals and the average "think time" (i.e., the average time between the system's response to a given transaction and the user's initiation of the next transaction). The principal performance factors calculated by BEST/1 for these workloads are the response time per transaction and the throughput rate in transactions per hour.

TRANSACTION PROCESSING WORKLOADS

Transaction processing workloads are similar to interactive workloads in that the external load is assumed to be generated by users at terminals. However, instead of specifying the external load in terms of number of terminals and think time, the analyst simply specifies a total arrival rate in "transactions per hour" as illustrated in Figure 4. Most analysts find it convenient to characterize transaction processing workloads in this manner.

Since the arrival rate is specified as an input parameter, and since throughput is equal to arrival rate (as long as system capacity is not exceeded), the most important performance value computed by BEST/1 for transaction processing workloads is the response time. BEST/1 also calculates the maximum attainable throughput rate in cases where this value is less than the arrival rate specified in the input file.

BATCH PROCESSING WORKLOADS

In the case of batch processing workloads, analysts are usually most interested in system performance during time periods when the input queues are backlogged (i.e., when

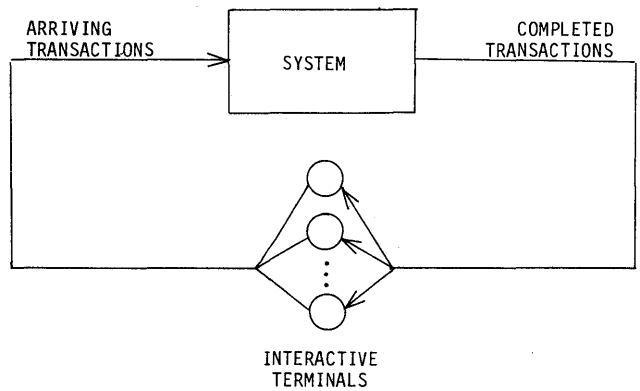


Figure 3—Time sharing workloads

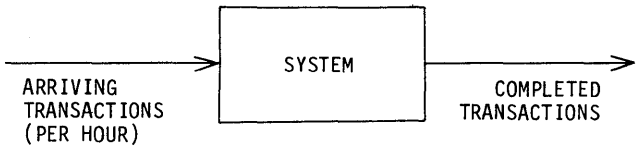


Figure 4—Transaction processing workloads

there is always at least one job waiting to be loaded into main memory). BEST/1 enables the analyst to specify batch workloads which have this "backlog" property. Schematically, these workloads are represented as shown in Figure 5. The closed loop in this figure indicates that, in a backlog situation, new jobs are loaded into main memory whenever an active job completes and memory space becomes available. Both throughput and response time (turnaround time) are computed for these workloads.

TOTAL NUMBER OF WORKLOADS

The total number of workloads which may be included in a single BEST/1 model is not limited to three. For example, a single BEST/1 model could include two transaction processing workloads, three batch processing workloads, and two time sharing workloads. Alternatively, a model could consist solely of one time sharing, one batch processing, or one transaction processing workload. In Release 4.0 the total number of workloads which may be included in a single BEST/1 model is equal to 9.

JOBS AND TRANSACTIONS

Each of the basic workload types discussed above makes use of the concept of a "job" or a "transaction". Essentially, these terms both refer to a unit of work which arrives at a system from an external source, undergoes a certain amount of processing, and is eventually completed. Throughput is expressed in jobs or transactions per hour, and response time (or turnaround time) is expressed on a per job or per transaction basis.

It is clear that the terms job and transaction refer to the same essential concept; however, the former term is more commonly used for batch processing workloads, whereas the latter is more common in the case of time sharing and transaction processing. In the following discussion of BEST/1, the two terms will be regarded as being interchangeable. However, the term "transaction" will be favored in most cases and should be understood to refer to either a batch job or an on-line transaction.

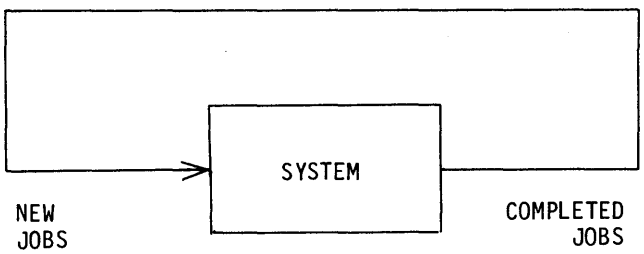


Figure 5—Batch processing workloads

SPECIAL WORKLOADS

By suitable selection and adjustment of parameters, the three basic workload types defined above can be used to represent a number of special workloads which are of importance in evaluating particular systems. These workloads include periodic real time requests, system overhead activities, and job streams with deadlines and critical paths.

TREATMENT OF MULTIPROGRAMMING

Figures 3-5 utilize a single box (SYSTEM) to represent the location of a transaction from the instant it arrives at a system to the instant it departs. The activity which takes place between these two points is described in greater detail in Figure 6.

The interpretation of Figure 6 is straightforward. Small rectangles indicate the locations of queues, and circles represent servers such as CPU's or I/O devices. As indicated in the diagram, an arriving transaction may at first have to wait in the memory queue until space becomes available. Once it is loaded into memory, the transaction is placed in the CPU queue where it waits until a processor becomes available. The transaction then receives a burst of CPU processing which terminates when an I/O request is generated. At this point, the transaction proceeds to the appropriate I/O device queue. When it reaches the head of the queue associated with that device, an I/O transfer is initiated. Upon completing this transfer, the transaction returns to the CPU queue. The alternating cycle of CPU bursts and I/O transfers continues until the processing requirements of the transaction are satisfied. The transaction then terminates and leaves the system via the "COMPLETED TRANSACTIONS" arrow.

BEST/1 enables the analyst to represent cases where several transactions are in main memory and active at the same

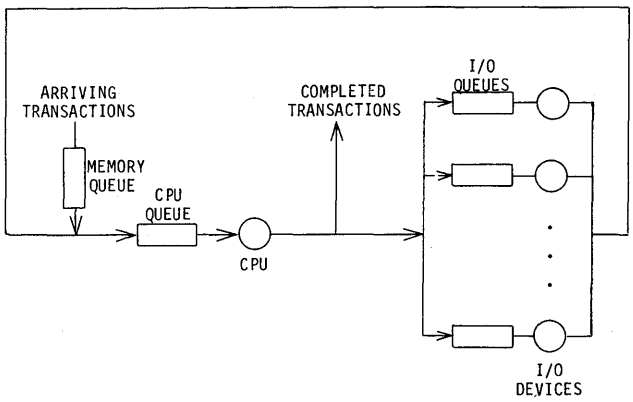


Figure 6—Multiprogramming activity

time. One transaction may be using the CPU while another is using an I/O device. Similarly, several I/O devices may be providing service to several different transactions at the same time.

This overlap of processing activity—which is generated by several transactions that are sharing the main memory of a single computer system—is generally referred to as multiprogramming. Multiprogramming tends to raise the utilization of system resources and thereby improve certain measures of overall performance. However, the queueing delays associated with multiprogramming can have an adverse effect on other measures of system performance. BEST/1 has the ability to explicitly represent the multiprogramming activity described above, and the analyst can use BEST/1 to determine the effectiveness of multiprogramming in various applications.

THE SERVER CONCEPT

As discussed in the preceding section, a job or transaction which has arrived at a system and has been loaded into main memory spends its time alternating between the following two states:

- Using active processing resources such as CPU's and I/O devices
- Waiting to use such resources

In the context of BEST/1, active processing resources are referred to as servers. That is, servers are points within a system where queues can form and where processing can be carried out. Each circle in Figure 6 corresponds to a different server.

One of the most important aspects of Figure 6 is the assumption that each server in the network is capable of independent operation. That is, it is assumed that the CPU can be processing one request at the same time that an I/O server is processing another request. Likewise, it is assumed that two or more I/O servers can be processing requests at the same time.

The assumption of simultaneous processing capability plays a critical role when the real I/O devices in a computer system are mapped into BEST/1 servers. For example, in the case of a string of high speed magnetic tape drives connected to a single controller and channel, it is only possible for one device to be actively transferring data at any time. Hence the entire string of tape drives should be represented as a single I/O server in Figure 6. On the other hand, if each tape drive were connected to a dedicated channel and controller, simultaneous operation would be possible and each device would be represented as a separate I/O server.

The issues that arise when representing disks and drums are somewhat more complex. In virtually all modern disk drives and controllers, seek overlap is possible. This means that any number of disks can be carrying out independent seek operations at the same time. Simultaneous overlap of the rotational latency time is also possible for disks and drums which employ rotational position sensing (RPS). However, the number of actual data transfers which can take place at the same time is limited to the number of parallel I/O channels that are connected to a particular controller and string of devices.

In cases such as these it is preferable to represent each disk and drum as a separate I/O server within BEST/1. That is, each disk and drum is assumed to be capable of independent simultaneous operation. The service time at each device must then be adjusted to account for the additional delays that arise because of channel contention. For example, disk service time—which is normally the sum of seek, rotational latency, and data transfer time—must be increased so that it becomes the sum of seek, rotational latency, data transfer, and channel queueing time. The increased service time represents a real effect since measurement devices would actually record the disk as being busy during channel queueing time.

BEST/1 provides facilities for calculating the amount of channel queueing that will arise within a particular configuration, and for automatically increasing I/O service times accordingly. Thus, channel contention is accounted for even though channels are not explicitly shown in Figure 6.

THE DOMAIN CONCEPT

As illustrated in Figure 7 the three principal components of a BEST/1 model are a set of workloads, a set of servers, and a set of memory domains. Essentially, memory domains are mechanisms for controlling the sharing of memory among various workloads. All multiple workload systems incorporate some mechanism of this type to prevent one or two workloads from monopolizing main memory in an uncontrolled manner. For example, time sharing transactions may be assigned to a domain with a maximum multiprogramming level of 5 while batch jobs are assigned to a domain with a maximum multiprogramming level of 7. This means that no more than 5 time sharing transactions and 7 batch jobs would be allowed to enter main memory at a

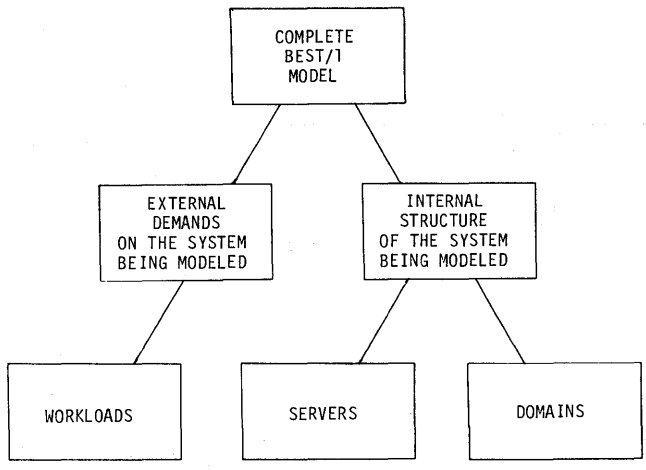


Figure 7—Components of a BEST/1 model

single time. Terms such as "partition", "region" and "initiator" are used in different operating systems to refer to the concepts of multiprogramming level and domain.

The specific mechanisms used to implement the memory domain concept vary somewhat from one system to another. BEST/1 allows a wide range of mechanisms to be modeled. Essentially, the analyst can characterize domains in terms of their maximum multiprogramming levels, their average multiprogramming levels, or in terms of certain distributions.

Note that memory domains are closely associated with workloads. In particular, each workload must be assigned to some domain. BEST/1 permits several workloads to be assigned to the same domain; alternatively, each workload may be assigned to a separate domain if desired.

As a final point, it should be clear from the preceding discussion that the memory queue shown in Figure 6 actually represents a set of queues, one corresponding to each domain. BEST/1 treats the queues separately, and values such as queue length and waiting time are computed individually for each memory queue.

BEST/1 INPUT PARAMETERS

Figure 8 presents an example of all the input data needed by BEST/1 for a typical application. In this case, the system being evaluated is processing three workloads: Workload 1 represents a stream of batch processing jobs, Workload 2 represents a transaction processing application, and Workload 3 represents the load generated by a set of time sharing users.

The BEST/1 input parameters used to characterize this application are grouped together on a "per workload" basis. For each workload, there are a set of workload descriptors and a set of service requirements. These two sets of parameters are discussed below.

WORKLOAD DESCRIPTORS

The workload descriptors in Figure 8 are used to specify the following information:

- The workload type: batch processing (BP), time sharing (TS), or transaction processing (TP)
- A label used to identify the workload on output reports
- Information about the multiprogramming level of the domain which is associated with the workload
- For time sharing and transaction processing workloads, information about the external factors which generate arriving transactions

In all cases, the first workload descriptor in the formatted BEST/1 input file contains the workload type and the workload label. The second descriptor characterizes the multiprogramming level of the workload's domain. For Workload 1, the average multiprogramming level attained during times when there is a backlog in the input queue is specified as

```
-----WORKLOAD 1-----DESCRIPTORS-----
          BP    BATCH PROCESSING
         3.8    AVERAGE MPL
-----WORKLOAD 2-----DESCRIPTORS-----
          TP    DATA BASE TRANS
         4.0    MAXIMUM MPL
      4000.0    TRANSACTIONS/HR
-----WORKLOAD 3-----DESCRIPTORS-----
          TS    TIME SHARING USERS
         6.0    MAXIMUM MPL
        30.0    NO. OF TERMINALS
        25.0    THINK TIME (SECS)
```

SERVER	WORKLOAD 1	WORKLOAD 2	WORKLOAD 3
1 CPU—370/168	13000.0	205.0	320.0
2 DRUM 1	1099.0	83.0	92.0
3 DRUM 2	1940.0	147.0	175.0
4 TAPE	395.0	0.0	0.0
5 TAPE	496.0	0.0	0.0
6 2314	747.0	0.0	0.0
7 2314	204.0	0.0	0.0
8 3330 SYSTEM PACK	4642.0	13.0	80.0
9 3330/ASPQ	12461.0	0.0	0.0
10 3330/ASPQ	12462.0	0.0	0.0
11 3330 SCRATCH	2474.0	64.0	42.0
12 3330 SCRATCH	2474.0	62.0	46.0
13 3330 SWAP AND USER	3034.0	150.0	100.0
14 3330 SWAP AND USER	3035.0	150.0	100.0
15 3330 MOD1 USER PACK	1073.0	27.0	200.0
16 3330 MOD1 USER PACK	1073.0	29.0	0.0
17 3330 MOD11 USER PACK	1727.0	49.0	0.0
18 3330 MOD11 USER PACK	1727.0	52.0	0.0

Figure 8—Basic input file

3.8. Workloads 2 and 3 are assigned maximum multiprogramming levels of 4 and 6 respectively.

Additional descriptors are used in Workloads 2 and 3 to characterize the external factors which generate arriving transactions. For Workload 2, these factors are characterized by specifying an average arrival rate of 4000 transactions per hour. For Workload 3, the external factors are specified as 30 terminals with average think times of 25 seconds.

SERVICE REQUIREMENTS

The second set of input parameters that are provided for each workload are referred to as the workload's service requirements. Essentially, these parameters specify the total amount of service time that is required, per transaction, at each server in the system. For example, each Workload 1 transaction (i.e., each batch job) requires an average of 13000 msec of processing at Server 1 (the CPU), 1099 msec of processing at Server 2 (Drum 1), 1940 msec of processing at Server 3 (Drum 2), and so on. The service requirements per transaction are specified separately for each workload as indicated in Figure 8.

Note that the service requirements in Figure 8 refer only to the time that a transaction spends actually using a server. The time spent waiting in a queue before reaching the server is not included in the input data, but is instead computed by BEST/1 and used to generate output reports.

EXTENDED EXAMPLE

BEST/1 enables the analyst to extend the basic system description of Figure 8 in a number of ways. These extensions are introduced by associating additional descriptors with each workload, and also by adding certain "special server parameters" to the end of the input file. Figure 9 provides an indication of the range of extensions which are available to the analyst.

```
-----WORKLOAD 1-----DESCRIPTORS-----
          BP    BATCH PROCESSING
         3.8    AVERAGE MPL
         3.0    PRIORITY
-----WORKLOAD 2-----DESCRIPTORS-----
          TP    DATA BASE INQUIRY
      3200.0    TRANSACTIONS/HR
         1.0    PRIORITY
         1.0    DOMAIN NUMBER
-----WORKLOAD 3-----DESCRIPTORS-----
          TS    TIME SHARING USERS
         6.0    MAXIMUM MPL
        30.0    NO. OF TERMINALS
        25.0    THINK TIME (SECS)
         2.0    PRIORITY
-----WORKLOAD 4-----DESCRIPTORS-----
          TP    DATA BASE UPDATE
       800.0    TRANSACTIONS/HR
         1.0    PRIORITY
         1.0    DOMAIN NUMBER
```

SERVER	WORKLOAD 1	WORKLOAD 2	WORKLOAD 3	WORKLOAD 4
1 CPU—370/168	13000.0	150.0	320.0	425.0
2 DRUM 1	1099.0	75.0	92.0	115.0
3 DRUM 2	1940.0	125.0	175.0	235.0
4 TAPE	395.0	0.0	0.0	0.0
5 TAPE	496.0	0.0	0.0	0.0
6 2314	747.0	0.0	0.0	0.0
7 2314	204.0	0.0	0.0	0.0
8 3330 SYSTEM PACK	4642.0	13.0	80.0	13.0
9 3330/ASPQ	12461.0	0.0	0.0	0.0
10 3330/ASPQ	12462.0	0.0	0.0	0.0
11 3330 SCRATCH	2474.0	20.0	42.0	240.0
12 3330 SCRATCH	2474.0	20.0	46.0	230.0
13 3330 SWAP AND USER	3034.0	100.0	100.0	350.0
14 3330 SWAP AND USER	3035.0	100.0	100.0	350.0
15 3330 MOD1 USER PACK	1073.0	15.0	200.0	75.0
16 3330 MOD1 USER PACK	1073.0	15.0	0.0	85.0
17 3330 MOD11 USER PACK	1727.0	20.0	0.0	165.0
18 3330 MOD11 USER PACK	1727.0	21.0	0.0	176.0

```
------DOMAIN  1----DESCRIPTORS-----
         LABEL    DATA BASE TRANS
           4.0    MAXIMUM MPL
------SERVER  1----DESCRIPTORS-----
         LABEL    CPU—370/168
           2.0    MULTIPLE SERVER
```

Figure 9—Extended input file

WORKLOAD DIFFERENTIATION

The most obvious difference between Figure 8 and Figure 9 is the fact that a fourth workload has been added. In this particular case, the additional workload is not intended to represent a new application being added to the system, although additional workloads could obviously be used for such purposes. Rather the new workload represents a more detailed model of the same system represented in Figure 8.

In the case of Figure 8 the transaction processing application is represented, using Workload 2, as a single input stream arriving at a rate of 4000 transactions per hour. BEST/1 will treat this workload as a single entity and will report response time, queue length, and other performance values for the workload as a whole.

Suppose, however, that the analyst now wishes to explicitly represent the fact that 80 percent of the Workload 2 input stream corresponds to data base inquiry transactions, and 20 percent corresponds to data base update transactions. In other words, suppose the analyst wishes to obtain separate performance values for each of the two transaction types.

In this case the analyst can use two separate workloads to represent the transaction processing application. As illustrated in Figure 9, Workload 2 is now used to represent the data base inquiry transactions. The arrival rate is set to 3200 transactions per hour (80 percent of 4000), and the service requirements are explicitly shown. In addition, Workload 4 is now used to represent the data base update transactions. The arrival rate is 800 transactions, and a different set of service requirements are provided. BEST/1 will then be able to compute response time, throughput, and other performance values for each transaction type.

LEVELS OF DETAIL

It should be emphasized at this point that the model represented by Figure 9 is not intrinsically "more correct" than the model represented by Figure 8. Both are "correct" in the sense that they both enable the analyst to investigate a certain set of questions concerning the performance of a real system. Since the model associated with Figure 9 is more detailed, a more detailed set of questions can be studied. However, the analyst is required to provide a more detailed set of input data in this case: specifically, the separate service requirements for each of the two transaction types.

Because of the ease with which the level of detail can be altered, BEST/1 enables the analyst to select an initial level of detail, run through a number of cases, and then increase the level of detail if necessary for those cases which warrant further analysis.

The ability to adjust the level of detail in a BEST/1 analysis is also important in applications involving capacity planning, design of new systems, and other situations where performance prediction is involved. In these cases the information available to the analyst tends to become more and more detailed as time progresses. BEST/1 thus enables the analyst to use the most detailed model that is feasible at each stage, and to increase the detail in the model as more refined information becomes available.

SHARED DOMAINS

Another important extension of Figure 8 which is illustrated in Figure 9 is the concept of the shared domain. It is implicitly assumed that each workload in Figure 8 has a separate domain and a separate memory queue. However, this is not the case for Workloads 2 and 4 in Figure 9. Rather, these two workloads are assumed to compete for the single domain which was assigned to Workload 2 in Figure 8.

The concept of two workloads sharing a single domain is easy to represent in BEST/1. As illustrated in Figure 9, this is done by placing a domain identifier rather than a multiprogramming level in the descriptors of Workloads 2 and 4. A separate domain descriptor is then added to the end of the input file to indicate the multiprogramming level associated with that shared domain. This is shown in Figure 9.

Note that the analyst is not required to use a shared domain in this case. For example, Workload 2 could have been assigned a maximum multiprogramming level of 3, and Workload 4 could have been assigned a maximum multiprogramming level of 1. This corresponds to a situation where each workload has a separate domain. BEST/1 will, of course, calculate a different set of performance values in this case. The analyst can then determine which memory allocation strategy is most appropriate in terms of the performance objectives of the original system.

PRIORITIES

BEST/1 permits the analyst to associate a CPU dispatching priority with each workload. In the case of Figure 8 no priorities were specified so BEST/1 assigned the same default priority to all workloads. Figure 9 illustrates the way priorities are added to the workload descriptors. As indicated in the figure, the transaction processing workloads have highest priority, the time sharing workload has intermediate priority, and the batch processing workload has lowest priority.

In cases where priorities are specified, BEST/1 uses the conventional "preemptive-resume" dispatching algorithm found in most operating systems. In other words, a high priority request which arrives at the CPU interrupts and preempts any lower priority requests that may be present. Lower priority requests resume processing from the point of preemption after higher priority requests release the CPU.

MULTIPROCESSORS

BEST/1 input files sometimes include "special server parameters" which refer to the servers within the system rather than individual workloads. The number of CPU's in a multiprocessor configuration is such a parameter. Figure

Principal Results Report
CPU and I/O Utilization Report
Response Time Profile
Average Queue Length Report
Average Number Waiting Report
Waiting Time Profile
Service Rate Degradation Report
Server Residency Profile
Queue Length Distribution Report
Concurrency Report
Memory Report
Throughput Report
Server Report
Waiting Time Percentages Report

Figure 10—BEST/1 output reports

9 illustrates a case where two CPU's are present. Special server parameters are used to represent other system features such as servers whose service rate varies as an arbitrary function of load.

GENERATION OF OUTPUT REPORTS

Once a particular set of input data has been specified, the BEST/1 user may request an evaluation. This will cause BEST/1 to analyze the data and generate the "Principal Results" report. After examining this report, the analyst may request one or more additional reports.

BEST/1 generates a number of different reports which provide detailed information about the performance of the system being evaluated. The report titles are listed in Figure 10.

PRINCIPAL RESULTS

Figure 11 illustrates the Principal Results report generated by BEST/1 from the input data shown in Figure 8. This report is organized by workload. For each workload, the report indicates the average response time per transactions, the average throughput in transactions per hour, and the CPU utilization associated with that workload. Total CPU utilization is also reported.

For example, in Figure 11 the batch processing throughput is 102 jobs per hour and the response times for transaction processing and time sharing are 3.17 seconds and 3.31 seconds respectively.

RESPONSE TIME PROFILE

Figure 12 illustrates the Response Time Profile report generated by BEST/1 from the input data shown in Figure 8. Essentially, this report gives the analyst a breakdown of the response time values that appear in the Principal Results report. That is, this report gives the response time, in milliseconds, at each server in the system. By adding up these individual response times, the total response time per transaction is obtained.

Note that main memory is represented in this report by "Server O". Thus, the response time at Server O corresponds to the delay a transaction experiences before being loaded into its memory domain. This delay only arises when the number of transactions at the system (for a given workload) exceeds the maximum multiprogramming level of the domain associated with that workload.

The Response Time Profile is particularly useful when it is necessary to determine why a workload's response time value is so high, and how important each server and queue really is insofar as overall response time is concerned.

INTERACTIVE DIALOG

The primary mode of operation for BEST/1 is through interactive time sharing terminals. When operating in this mode, the analyst has access to a variety of functions. These functions are requested through an interactive dialog that has been extensively refined and enhanced during several years of actual use.

The complete interactive dialog is specified in the BEST/1 User's Guide. Some typical functions which are easy to carry out using the dialog are listed below:

- Add servers or workloads
- Change server speed
- Change arrival rate, number of terminals, CPU dispatching priority, multiprogramming level, etc.
- Request one or more optional reports
- Save an input file for future use
- Direct output to any desired device
- Execute a pre-defined set of interactive commands stored in a file

It should be emphasized that this is only a partial list of the functionality available to the BEST/1 user.

In all cases, the interactive dialog has been structured using concepts that can be easily and directly understood by

♦♦♦PRINCIPAL RESULTS♦♦♦

WORKLOAD	RESPONSE TIME	THROUGHPUT	% CPU
1 BATCH PROCESSING	133.98 SEC	102. PER HOUR	36.9%
2 DATA BASE TRANS	3.17 SEC	4000. PER HOUR	22.8%
3 TIME SHARING USERS	3.31 SEC	3815. PER HOUR	33.9%
		TOTAL CPU UTILIZATION =	93.6%

Figure 11—Principal results report

	RESPONSE TIME PROFILE BY WORKLOAD		
SERVER	1	2	3
0 MEMORY	0.0	838.1	221.5
1 CPU—370/168	65680.5	1159.7	1873.0
2 DRUM 1	1410.7	107.0	119.4
3 DRUM 2	3244.4	242.0	292.4
4 TAPE	397.4	0.0	0.0
5 TAPE	500.2	0.0	0.0
6 2314	757.8	0.0	0.0
7 2314	204.3	0.0	0.0
8 3330 SYSTEM PACK	5865.2	17.1	105.6
9 3330/ASPQ	17053.1	0.0	0.0
10 3330/ASPQ	17054.9	0.0	0.0
11 3330 SCRATCH	3005.0	79.5	52.3
12 3330 SCRATCH	3012.5	77.2	57.4
13 3330 SWAP AND USER	4675.4	231.8	156.4
14 3330 SWAP AND USER	4677.1	231.8	156.4
15 3330 MOD1 USER PACK	1477.6	37.3	277.6
16 3330 MOD1 USER PACK	1137.1	31.4	0.0
17 3330 MOD11 USER PACK	1909.1	55.4	0.0
18 3330 MOD11 USER PACK	1916.5	59.0	0.0
TOTAL RESPONSE TIME	133978.6	3167.1	3312.1

Figure 12—Response time profile

the computer performance analyst. Knowledge of the underlying mathematical theory is not required to set up BEST/1 models or exercise the interactive dialog.

SUBROUTINE INTERFACE

Users are sometimes interested in carrying out complex iterative analysis procedures which repeatedly invoke BEST/1. For example, a user may wish to increase a workload's arrival rate until the resulting response time reaches some threshold.

Some cases can, of course, be treated using the BEST/1 interactive dialog. However, it is sometimes preferable to simply write a program which repeatedly calls BEST/1, analyzes the output, and adjusts certain BEST/1 input parameters until a condition has been satisfied or a set of cases has been examined.

In order to facilitate this process, there is a special interface which allows BEST/1 to be called as a subroutine by a FORTRAN program. The FORTRAN program can then be executed interactively or in batch processing mode.

ACKNOWLEDGMENTS

Special thanks are due to Professors Peter J. Denning and Stuart E. Madnick and to Dr. Gary Sockut for their helpful suggestions and comments regarding the design of BEST/1.

BIBLIOGRAPHY

1. Bard, Y., "An Analytic Model of CP-67 and VM/370," *Computer Architectures and Networks,* North-Holland, 1974, pp. 419-460.
2. Baskett, F. et al, "Open, Closed and Mixed Network of Queues with Different Classes of Customers," *JACM,* Vol. 22, No. 2, April 1975, pp. 248-260.
3. Brandwajn, A., "A Model of a Time Sharing Virtual Memory System Solved Using Equivalence and Decomposition Methods," *Acta Inf.,* Vol. 4, No. 1, 1974, pp. 11-47.
4. Buzen, J. P., "Queueing Network Models of Multiprogramming," Ph.D. Thesis, Harvard University, May 1971, Report AD 731 575, NTIS, Springfield, Virginia, August 1971.
5. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Comm. ACM,* Vol. 16, No. 9, Sept. 1973, pp. 527-531.
6. Buzen, J. P. and P. S. Goldberg, "Guidelines for the Use of Infinite Source Queueing Models in the Analysis of Computer System Performance," *AFIPS Conference Proceedings,* Vol. 43, 1974, pp. 371-375.
7. Buzen, J. P. and A. W. Shum, "Structural Considerations for Computer System Models," *Proc. Eighth Princeton Conference on Information Sciences and Systems,* March 1974, pp. 335-339.
8. Buzen, J. P., "Fundamental Operational Laws of Computer System Performance," *Acta Inf.,* Vol. 7, No. 2, 1976, pp. 167-182.
9. Buzen, J. P., "Operational Analysis: The Key to the New Generation of Performance Prediction Tools," *Proc. IEEE COMPCON,* September 1976, pp. 166-171.
10. Courtois, P. J., "Decomposability," *ACM Monograph Series,* Academic Press, 1977.
11. Denning, P. J. and Buzen, J. P., "Operational Analysis of Queueing Networks," *Proc. Third Int'l. Symposium on Modeling and Performance Evaluation of Computer Systems,* North-Holland, October, 1977, pp. 151-172.
12. Giammo, T. P., "Extensions to Exponential Queueing Network Theory for Use in a Planning Environment," *Proc. IEEE COMPCON,* September 1976, pp. 172-177.
13. Gordon, W. J. and G. F. Newell, "Closed Queueing Systems with Exponential Servers," *Oper. Res.,* Vol. 15, No. 2, 1967, pp. 254-265.
14. Jackson, J. R., "Jobshop-Like Queueing Systems," *Management Science,* Vol. 10, No. 1, 1963, pp. 131-142.
15. Lazowska, E. D., "The Use of Percentiles in Modeling CPU Service Time Distributions," *Proc. IFIP 1977 Int'l. Symposium on Computer Performance Modeling, Measurement and Evaluation,* August 1977, pp. 53-66.
16. Moore, C. G., "Network Models for Large-Scale Time-Sharing Systems," Ph.D. Thesis, University of Michigan, April 1971.
17. Reiser, M. and H. Kobayashi, "On the Convolution Algorithm for Separable Queueing Networks," *Proc. IFIP/ACM Symposium on Computer Performance Modeling, Measurement and Evaluation,* March 1976, pp. 109-117.
18. Rose, C. A., "A Calibration-Prediction Technique for Estimating Computer Performance," *AFIPS Conference Proceedings,* Vol. 46, 1977, pp. 813-818.
19. Sevcik, K., et al, "Improving Approximations of Aggregated Subsystems," *Proc. IFIP 1977 Int'l. Symposium on Computer Performance Modeling, Measurement and Evaluation,* August 1977, pp. 1-22.
20. Shum, A. W., "Queueing Models for Computer Systems with General Service Time Distributions," Ph.D. Thesis, Division of Engineering and Applied Physics, Harvard University, December 1976.
21. Shum, A. W. and J. P. Buzen, "The EPF Technique: A Method for Obtaining Approximate Solutions to Closed Queueing Networks with General Service Times," *Proc. Third Int'l. Symposium on Modeling and Performance Evaluation of Computer Systems,* North-Holland, October 1977, pp. 201-220.
22. Williams, A. C. and R. A. Bhandiwad, "A Generating Function Approach to Queueing Network Analysis of Multiprogrammed Computers," *Networks,* Vol. 6, No. 1, January 1976, pp. 1-22.

Job scripts—A workload description based on system event data*

by ROBERT L. MEAD and HERBERT D. SCHWETMAN
Purdue University
West Lafayette, Indiana

INTRODUCTION

Job scripts, a detailed description of a job's resource demands, can serve not only as input for a simulation model, but as a representation of the system's workload as well. Heretofore, most representation efforts have dealt with either benchmark or synthetic job techniques. This paper presents a workload representation based on a set of job scripts, each representing a single job in the workload, that is designed for use with simulation models. This representation differs from the synthetic job approach in the form of the representation, the intended usage, and the level of detailed involved. The scripts have been used as input for a trace-driven simulation model of the CDC6500 system at Purdue University and have also been used to increase the understanding of the execution time characteristics of programs.

The accurate representation of the workload is an important problem in the modeling and evaluation of computer systems. In 1972, Grenander and Tsao[1] stated, "We believe that no real significant advance in the evaluation of systems can be expected until some breakthrough is made in the characterization of the workload." Since that time, advances have been made in understanding program behavior in terms of memory reference behavior,[2–6] and some work has appeared on CPU usage behavior.[7] Job scripts can be used to increase the understanding of how programs utilize CPUs, data channels, and I/O processors. One outgrowth of this investigation has been *activity plots* which depict a program's file and device usage patterns, in much the same way that page reference maps show the pattern of memory demands.

The paper begins by summarizing earlier efforts in workload representation. The form of the job script and the generation techniques are presented. The issues of accuracy and the reproducibility of the job script are discussed, and data is presented to demonstrate that the scripts are reproducible and accurate. The advantages, limitations, and usefulness of job scripts are presented, along with some examples of the activity plots mentioned previously.

* This work was supported in part by NSF grant GJ-41289.

PREVIOUS WORK

As early as 1965 it was apparent that carefully choosing a representative workload was important when evaluating a computer system. Joslin[8,9] stressed the importance of knowing the nature of a workload to be represented by a benchmark and demonstrated that the performance of a system could vary depending upon the benchmark chosen. DeMeis and Weizer[10] represented interactive programs with *pseudo-programs* that were built around a loop of clocking instructions. Also in 1969, Buchholz[11] first introduced the idea of a *synthetic program,* a parameterized cyclic program that performed no useful work, but that was designed to consume a desired amount of CPU and I/O time. Wood and Forman[12] used accounting data and the output from a hardware monitor to parameterize synthetic programs in order to test the throughput of IBM 360 configurations. Synthetic programs have also been used in a number of other efforts. Schwetman and Browne[13] used an event driven software monitor to provide parameters for a synthetic job stream generator, and Kernighan and Hamilton[14] successfully used synthetic jobs to represent individual jobs in a standard benchmark. Sreenivasan and Kleinman[15] describe a technique for generating a synthetic workload by matching the joint probability densities of the real and synthetic workloads. Agrawala and Mohr[16] present a general model involving mixture distributions in a chosen n-space, leading to a representation that can be validated statistically. Cohen[17] described a *job catalog* that can be used to describe a benchmark set of programs.

The concept of workload representation in the form of trace data for trace-driven simulation models was proposed by Cheng[18] in 1969. Sherman[19] and Sherman et al.[20] successfully used a trace-driven model to investigate deadlock algorithms and CPU scheduling strategies. A forerunner to the job script was presented by Keim and Schwetman,[21] and was used to produce statistical descriptions of CPU and system routine usage intervals.

THE JOB SCRIPT AS A WORKLOAD REPRESENTATION

The basic concept underlying the job script is that of a *CPU—I/O cycle.* As a job executes, it typically uses the

CPU for a period of time and then requests an I/O operation. At the completion of the I/O operation, or immediately after the issuance of the I/O request in the case of overlapped I/O activity, the CPU is again used until a subsequent I/O request. This pattern of behavior continues until the job completes.

A job script represents this behavior in terms of an extended sequence of alternating CPU intervals and I/O sequences. CPU intervals represent the total amount of CPU time actually consumed between successive I/O requests and the I/O sequences consist of the resource usage intervals involved in I/O operations: I/O processor intervals and data transfer intervals. The total resource usage of a job is thus represented by CPU, I/O processor, and data channel usage intervals. It is important to note that we are *not* dealing with a virtual memory system. The job script representation assumes that all of the memory a job will need at a given time is allocated to the job in a contiguous block. It is possible however, for an executing job to request additional memory, or to release memory (at the end of the block) that is no longer needed.

Additional detail concerning the execution time characteristics of jobs is also included by the insertion of information entries in the script. These entries provide information about the type of I/O operation requested and the associated file name, the number of characters transferred during an I/O operation, the name of the I/O processor routine used, the current amount of memory being held by a job, and the name of the current job step (control card). This combination of resource usage intervals and information entries provides a complete and detailed description of the demands placed on the system by an individual job.

Ideally, the job scripts form a load independent representation, since the time a job spends waiting for resources is eliminated. In practice however, the processing load does affect the length of some resource usage intervals to a small degree, due to such things as varying seek times and rotational delays for disk accesses, and varying amounts of system overhead. The effect of the load on the script, along with the issues of accuracy, reproducibility, and usefulness of job scripts are discussed in subsequent sections of this paper.

The job script approach differs from the synthetic job approach on one key point. The script technique is essentially a bottom-up approach, where every system event of interest is examined in order to create the scripts. This differs from the formation of synthetic jobs where information with little detail (e.g., accounting data) is used as a starting point, and the detail required to make the job realistic is introduced. It is important to note that the data from which the scripts are generated is gathered at the level at which a job interacts with the system. This implies that, as long as we restrict ourselves to monitoring techniques that are external to the job itself, we cannot increase the level of detail included in the representation.

JOB SCRIPT GENERATION

The generation of the job scripts involves the use of a software monitor and two analysis programs. A software system monitor first collects event data while the system is in operation. The event data is processed to produce a temporary file consisting of all requests, assignments, and releases of system resources for each job, plus all of the additional information to be included in the job script. The temporary file is processed to eliminate all queueing delays experienced by each job and to remove any activity solely concerned with load dependent activities. In the system being studied, it is possible for a job to be preempted from memory prior to completion and then to be returned to memory at a later time. Since we are not dealing with a virtual memory system, when a job is to be preempted from memory, it is necessary to save the contents of the job's entire memory allocation, plus the contents of all hardware registers associated with the job. Thus both the preemption and subsequent resumption require I/O processors and data channels to copy the contents of memory to and from a disk. The number of preemptions incurred by a job depends on the system load, and as such they are not included in the job script. These preemptions and resumptions represent the majority of the load dependent activity eliminated from the job scripts.

It should be noted that the removal of this load dependent behavior does not imply that this activity is ignored in the simulation model driven by the scripts. The activity associated with job preemption and resumption is included in the trace-driven simulation model but is generated from probability distributions when the simulation model decides a job preemption or resumption should take place. The elimination of this load dependent behavior from the script representation thus allows for greater flexibility in the simulation model.

The details of the software monitor can be found in References 22 and 23, and a modified version of the data analyzer described in References 23 and 24 is used to generate the temporary file. The basic problem involved in the reduction of the data on the temporary file is the recognition of the resource usage intervals. This problem is complicated by the presence of overlap between CPU and I/O activity and by the fact that several distinct I/O sequences may be in progress simultaneously. As will be seen in the example below, this implies that the order of the intervals in the job script is not necessarily the order of occurrence during ex-

Event Number	Time	Event	Resource	Event Number	Time	Event	Resource
1	0	Req	CPU	16	32	Req	CH01 (3)
2	1	Ass	CPU	17	40	Ass	CH01 (3)
3	4	Rel	CPU	18	43	Req	CPU
4	4	Req	CPU	19	43	Ass	CPU
5	10	Ass	CPU	20	46	Rel	CPU
6	13	Req	IOP	21	55	Rel	CH01 (3)
7	15	Ass	IOP 5	22	56	Req	CPU
8	17	Rel	CPU	23	58	Ass	CPU
9	19	Req	CH00 (5)[1]	24	59	Rel	IOP 3
10	20	Req	CPU	25	61	Rel	CPU
11	22	Ass	CPU	26	63	Rel	CH00 (5)
12	25	Ass	CH00 (5)	27	64	Rel	IOP 5
13	27	Req	IOP	28	67	Req	CPU
14	28	Ass	IOP 3	29	67	Ass	CPU
15	29	Rel	CPU	30	70	Req	IOP

[1] I/O processor number in parenthesis

Figure 1—Example of event records in the intermediate file

Script Entry	Resource	Interval Length	Interval Bounds[1] Start	End
1	CPU	6	2	6
2	IOP 5	4	7	9
3	CH00	38	12	26
4	IOP 5	1	26	27
5	CPU	9	6	13
6	IOP 3	4	14	16
7	CH01	15	17	21
8	IOP 3	4	21	24
9	CPU	11	13	30

[1] Start and End denote event records in Figure 1 that define the interval

Figure 2—Example of a job script

ecution. Figure 1 contains an example of the event records that might appear in the temporary file for a portion of a job. The job script entries subsequently generated appear in Figure 2. Note that the intervals corresponding to script entries 5, 6, 7, and 8 all occur before the interval corresponding to entry 4. The order in the script does, however, correspond to the sequence that would result if overlap of I/O processors did not take place.

ACCURACY AND REPRODUCIBILITY OF JOB SCRIPTS

In order to be of benefit, the scripts must be shown to be both an accurate and reproducible representation of a job's behavior. The correctness of the script was determined by listing the event records from which the scripts are produced and verifying that the resultant script accurately interprets the event records. Reproducibility is more difficult to verify, mainly because of variation in resource usage that is not under the control of the script generation process. Two major reasons for variations of this nature are varying seek times and rotational delays in data channel intervals and varying amounts of system overhead included in CPU usage.

As the monitor is unable to measure the seek time for disk accesses, the placement of files on the system disks and the load on the system can cause access times to vary greatly for different executions of the same job. This leads to fluctuations in the scripts. Rotational delay times also vary, but over time (assuming we are observing a lengthy job) these variations should tend to average out. In addition, the monitor does not record all CPU changes to supervisor mode, so the length of recorded CPU intervals can vary depending on the amount of system overhead occurring while a job is using the CPU.

To ascertain the degree of variation in the job scripts, four jobs of differing characteristics were each executed four times in a monoprogrammed environment. Scripts were generated for each of these jobs and analyzed using a statistics gathering program. For each job script the following quantities were determined:

(1) memory residency time
(2) adjusted memory residency time (memory residency time minus the time during which only load dependent activity occurred)
(3) total work of the job (CPU, I/O processor, and data channel time)
(4) total CPU time
(5) total I/O processor time
(6) total data channel time
(7) total number of I/O transfer units (1 transfer unit = 640 characters)
(8) total number of I/O sequences

Table I contains the statistics compiled for one copy of each of the four jobs. A maximum relative error for each quantity for each job was determined by using the smallest observed value of each quantity as the correct value. As can be seen from the summary of results in Table II, all of the measured quantities are reasonably consistent, with the exception of the data channel times. This is not surprising, however, due to the problems of seek times and rotational delay discussed above. We have attempted to determine the reason for the rather high degree of variation in the data channel time for job 2 (20.6%), but the difference seems to be caused only by the inherent variation within the system.

It is not usually possible to execute jobs in a monoprogrammed environment, however, so the script must be able to eliminate variation caused by the load in a multiprogrammed environment. To determine the effects of multiprogramming, the same four jobs were executed four times in a multiprogrammed environment. Job scripts were again generated, and the same set of statistics collected. The maximum relative errors determined for the multiprogrammed jobs appear in Table III. Once again most of the measurements are consistent. Memory residency times fluctuate to a greater extent, but the fact that the total work of each job

TABLE I.—Sample Statistics of Monoprogrammed Jobs

	Job 1	Job 2	Job 3	Job 4
Memory Residency Time[1]	8.321	36.360	52.169	14.864
Adj. Memory Res. Time[1]	7.752	35.099	51.353	14.298
Total Work[1]	8.008	35.895	62.470	15.867
CPU Time[1]	1.810	18.899	26.060	9.447
I/O Processor Time[1]	1.338	1.552	3.027	0.873
Data Channel Time[1]	4.860	15.444	33.383	5.547
No. I/O Sequences	183	268	762	151
Total Transfer Counts[2]	814	1461	2358	1115

[1] seconds
[2] 1 transfer count = 640 characters transferred

TABLE II—Maximum Relative Errors over Four Runs Monoprogrammed Jobs

	Job 1	Job 2	Job 3	Job 4
Memory Residency Time	5.0%	7.6%	3.8%	3.9%
Adj. Memory Res. Time	5.2	8.3	3.7	3.8
Total Work	6.0	8.2	2.3	4.7
CPU Time	1.5	0.2	0.7	0.3
I/O Processor Time	1.3	2.1	2.8	1.1
Data Channel Time	9.7	20.6	4.1	12.6
No. I/O Sequences	0.0	0.7	2.1	1.3
Total Transfer Counts	0.0	0.0	0.1	0.1

TABLE III—Maximum Relative Errors over Four Runs
Monoprogrammed Jobs

	Job 1	Job 2	Job 3	Job 4
Memory Residency Time	22.1%	7.3%	11.5%	31.7%
Adj. Memory Res. Time	16.0	9.9	9.7	37.4
Total Work	6.7	2.8	6.6	10.6
CPU Time	6.4	1.1	1.6	3.3
I/O Processor Time	10.7	6.7	4.9	18.9
Data Channel Time	8.6	7.5	10.8	20.0
No. I/O Sequences	4.4	1.5	0.5	1.4
Total Transfer Counts	0.4	0.0	0.1	0.0

TABLE V—Comparison of Totals
Multiprogramming over Monoprogramming

	Mono-programmed	Multi-programmed	Increase
Memory Residency Time	449.371	577.242	28.5%
Adj. Memory Res. Time	436.078	535.144	22.7
Total Work	490.942	516.476	5.2
CPU Time	225.562	232.270	3.0
I/O Processor Time	27.092	28.824	6.4
Data Channel Time	238.468	253.344	6.2
No. I/O Sequences	5440	5431	-0.2
Total Transfer Counts	22,991	22,999	0.0

exhibits a smaller variation (relative to the memory residency times) indicates that a majority of the effects of multiprogramming have been eliminated.

Finally, the job scripts generated in the monoprogrammed and multiprogrammed environments were compared. The average value of each measure for each job was determined in both environments, and the percentage increase caused by the multiprogramming, relative to the monoprogrammed value, was computed (i.e., (multi-mono)/mono). The results of these comparisons appear in Table IV. In addition, the sum of each quantity over all 16 jobs was determined for each environment, and the increase in the multiprogrammed environment determined. These results appear in Table V. It seems reasonable to expect a slight increase in observed resource usage due to increased overhead and seek times in a multiprogrammed environment, and the data confirms this expectation.

Based on the data in Tables II-V, the job scripts appear to be reproducible and relatively insensitive to the load on the system. Those variations that are present seem to be inherent to the system itself, and as such, any workload characterization effort would be unable to prevent such variations from being included in the workload description.

ADVANTAGES AND USES OF THE JOB SCRIPT

The primary advantage of the job script as a workload representation lies with its completeness. The scripts contain a wealth of detail, and they can be analyzed in a number of different ways. File usage patterns can be studied, including the types of operations and the number of characters transferred in each operation. The characteristics of each job step can be investigated by using the job step name entries. Since each resource usage interval is included, the distribution of device service time intervals can be displayed in addition to the determination of the mean and standard deviation of the distributions.

TABLE IV.—Average Percentage Increase Multiprogrammed over Monoprogrammed

	Job 1	Job 2	Job 3	Job 4
Memory Residency Time	64.2%	42.8%	4.5%	58.0%
Adj. Memory Res. Time	50.5	35.8	3.9	44.9
Total Work	4.6	7.7	1.3	15.6
CPU Time	7.9	3.1	1.7	5.5
I/O Processor Time	10.0	13.4	0.6	8.3
Data Channel Time	1.9	13.0	-0.5	32.3
No. I/O Sequences	1.5	0.3	-0.7	-0.3
Total Transfer Counts	-0.2	0.3	-0.0	-0.0

One such use of the job scripts has been the generation of *activity plots,* indicating the patterns of file and device usage. Activity plots are similar in nature to page reference maps that depict a job's memory demands.[5,25,26] Virtual time is displayed along the horizontal axis, and the various files and devices used by the job appear on the vertical axis. The time axis is divided into time intervals, and a vertical line is drawn in the interval for each file or device in use during the interval. Figures 3, 4, and 5 are examples of activity plots generated from the job scripts. The three entries labeled "IOP ROU" are used for I/O sequences that do not contain file name entries and usually represent some supervisory function. The entry labeled "1AJ/1CJ" represents the I/O processor routines that advance a job to the next job step and handle job completion. The entry labeled "PFILES" is marked for any I/O sequence that uses a file associated with the permanent file system. The remaining names appearing on the Y axis are either system files or user files used by the job. The plots in Figures 3 and 4 are of the same job, executed in monoprogrammed and multiprogrammed environments respectively. These activity plots offer visual evidence of the reproducibility of job scripts, as the two plots appear to be nearly identical. The activity plot in Figure 5 indicates that jobs may pass through several *phases*, with cyclic behavior within phases. This type of behavior appears to be analogous to the phase/transition behavior of several memory reference models being proposed.[26] Additional work is planned to investigate in more detail this phase behavior with respect to file and device usage.

In addition to providing a means for investigating a job's execution characteristics, the job scripts are also suitable for use in trace-driven simulation models.[19,20] A fairly detailed model of our system has been implemented using the scripts as trace data to drive the model. In order to validate the simulation model and the use of the job script representation, a test workload of 16 jobs was selected. This workload was executed on the CDC6500 system at Purdue, and job scripts were generated. Measurements of system performance were also taken using the software monitor and compared with the results from the simulator. Three quantities were chosen for validation: completion time of the entire workload, mean adjusted memory residency time, and

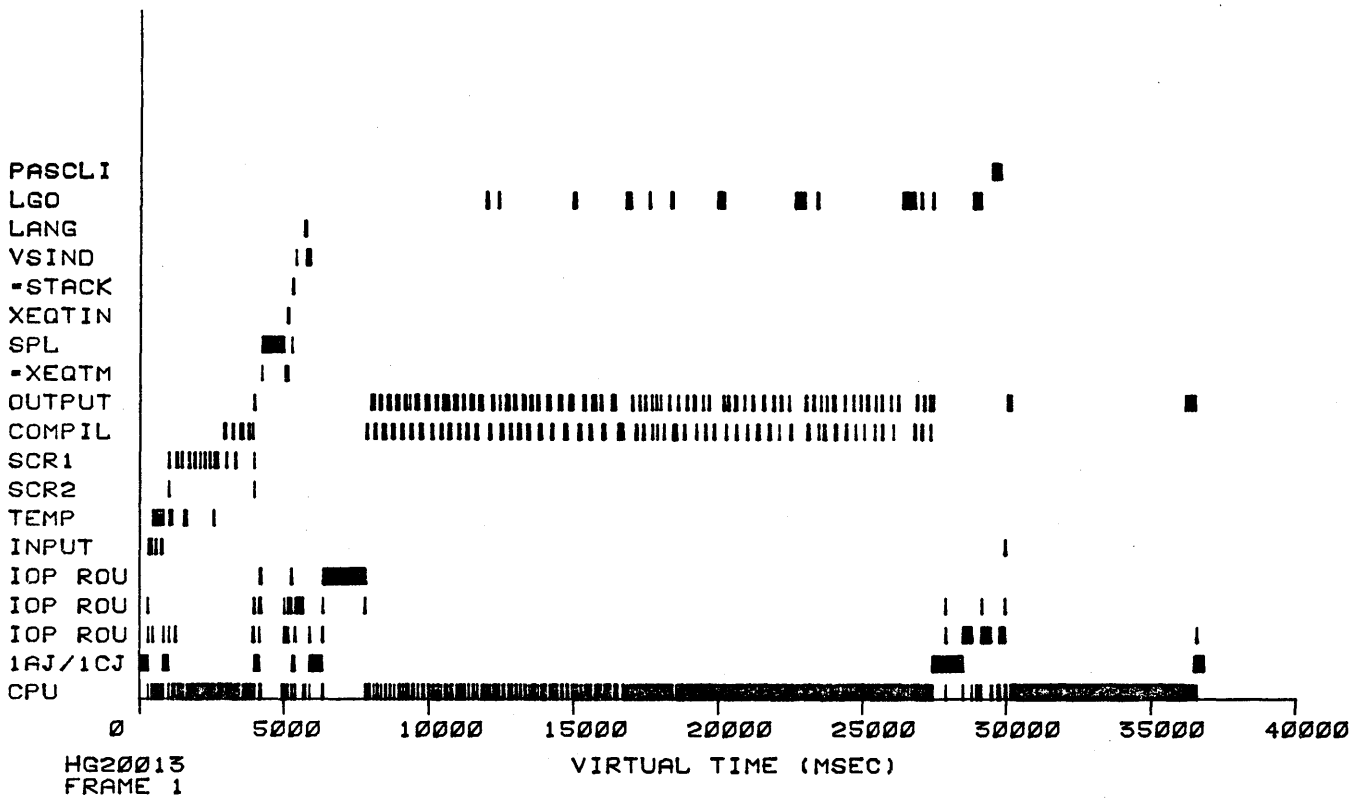


Figure 3—Activity plot of monoprogrammed job

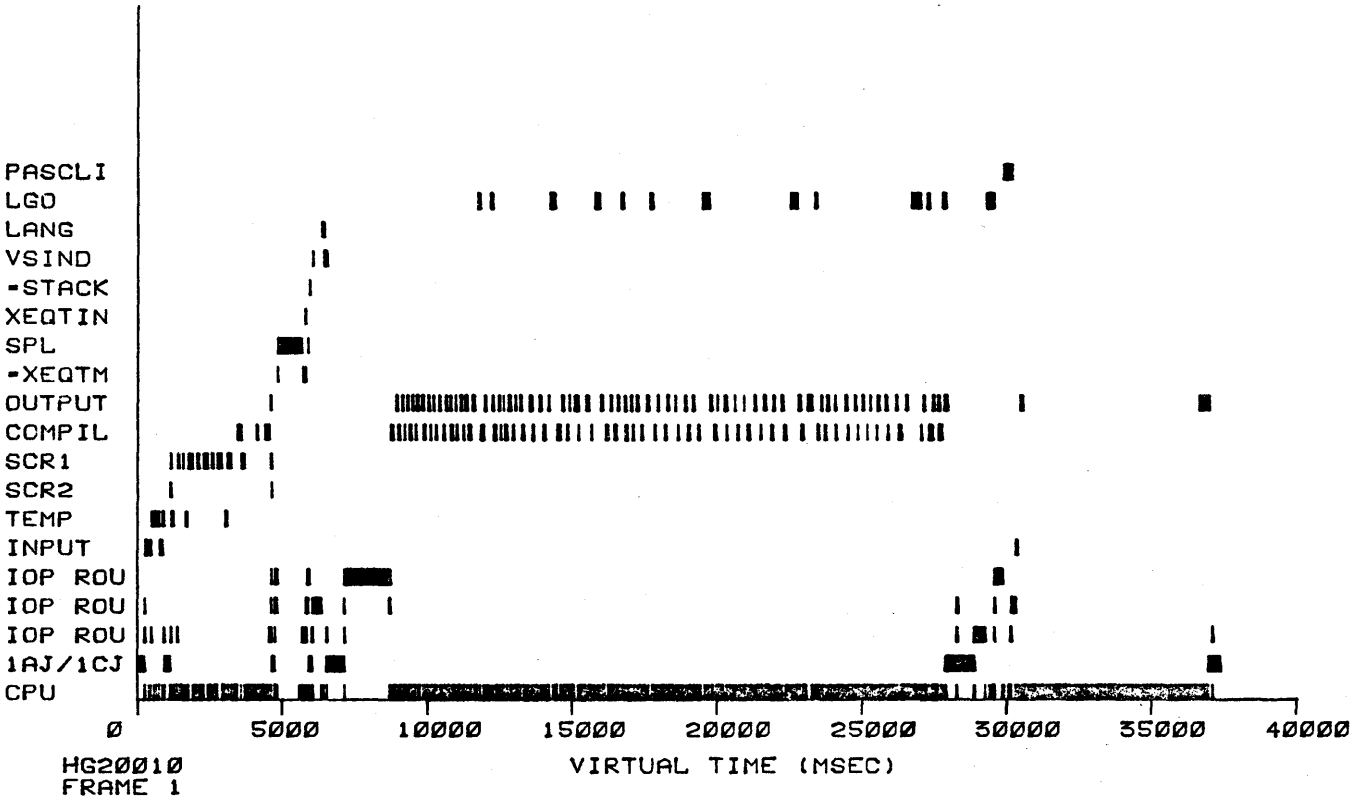


Figure 4—Activity plot of multiprogrammed job

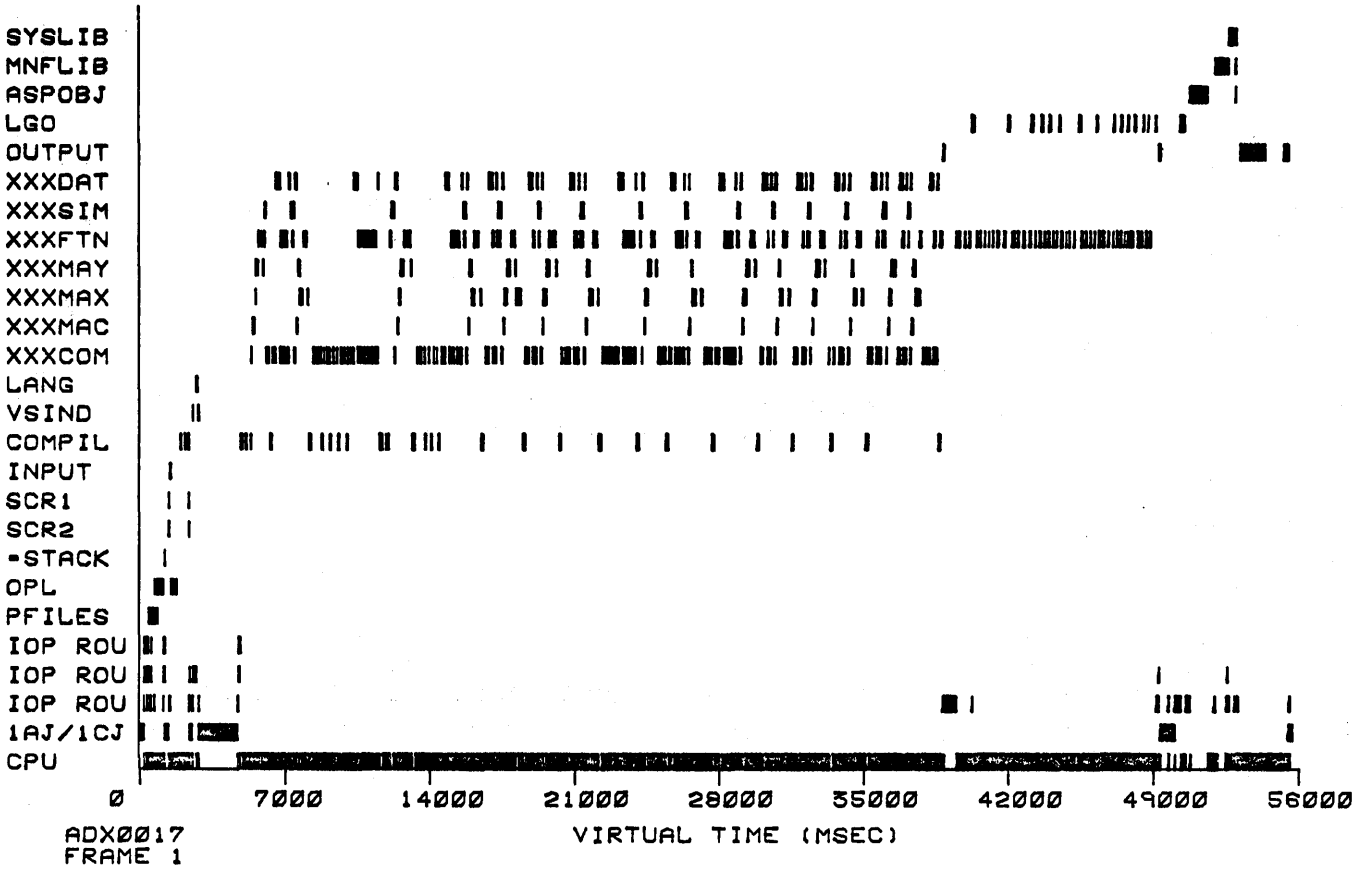


Figure 5—Activity plot showing phase behavior

mean response time. Since all of the jobs were available for execution at the start of the experiment, the mean response time is just the average of the individual job completion times. It was not possible to duplicate in the simulator the exact scheduling algorithm used by the real system, so each job in the model was given an arrival time that corresponded to the time the job first started execution in the real system. The job script accurately reflects the resource demand for each device in the system, so device utilizations were not used for validation, since accurate modeling of the completion time implies accurate modeling of the device utilizations.

In the initial test of the model, relative errors of 4.3 percent, 6.4 percent, and 10.6 percent were observed for the completion time, mean residency, and mean response times respectively. Subsequently the model was modified to use the file name and I/O operation entries to allow for CPU-I/O overlap on WRITE operations. After the modification was complete, the relative errors were 1.0 percent, 3.1 percent, and 6.7 percent. A summary of these results appears in Table VI. The number of job preemptions in each case is also included in the table, and it appears that the model does well in this respect also. Thus the job script representation can be used in a simulation model to approximate relatively closely the overlapped behavior in the real system. Further work is also planned on the use of the job script with trace-driven simulation models.

Finally, the job scripts can also be used to determine the parameters for a central server model.[27] This is possible due to the basic structure of the job script, the CPU-I/O cycle. The transition probabilities are easily determined from a count of the number of intervals for each device, and the mean service times are easily obtained. We plan to produce a sequence of central server model parameters for a job, each set of parameters corresponding to a phase of the job. In combination with the memory requirement entries in the job script, it is hoped that these parameters can be used in hybrid simulation models[28] as well.

TABLE VI.—Validation of Simulation Model

	Actual System	Simulation Model	
		Without Overlap	With Overlap
Completion time[1]	119.886	125.101	118.709
Mean residency time[1]	24.439	26.006	25.197
Mean response time[1]	53.295	58.922	56.869
Job preemptions	13	10	11

[1] Seconds

LIMITATIONS OF THE JOB SCRIPT

The use of job scripts as a workload representation has some drawbacks. The major disadvantages are the need of a software monitor, the expense of producing the job scripts, and the lack of compactness in the representation.

The use of a software monitor to gather the initial event data is an obvious limitation. The usual disadvantages associated with the use of a software monitor are also implicitly involved. Detailed knowledge of the operating system is required for the correct installation, operation, and maintenance of the monitor. The monitor must also be modified whenever changes are made to the operating system. A program is required to analyze the event data produced by the monitor, and in the case of script generation, it must also produce the input required by the script generation program.

The generation of the job script is also a somewhat expensive process in terms of the computer time involved. Table VII lists the cost of each step in the script generation process for the 16 jobs executed in a monoprogrammed environment. Step 4, script analysis, is not part of the script generation process per se, but does provide detailed statistics about the scripts generated. The costs involved are higher when scripts are generated from data collected from the production system over a 20 minute period (in excess of $75.00 and over 1300 CPU seconds). Fortunately, the script generation process is a one time cost, and the scripts produced can be used in a variety of ways.

Another potential limitation of the job script representation is that it is essentially just a record of job activity as observed by the system. For example, the job script's perception of the I/O behavior of a program is obscured by the buffering of logical I/O activity (i.e., the software monitor detects I/O activity only when the buffer of the associated file is either empty or full). It is the authors' contention however, that unless we are willing to instrument each program individually (or instrument the system I/O routines), we must be content with this "distorted" view of job behavior. We also feel that since the job script is produced from information gathered at the point where a program interacts with the system, it should be a sufficient representation for most applications. The job script representation as described appears to be suitable for use only with non-virtual memory systems. It is possible however that it could be adopted for use with a paged virtual system. At the very minimum, additional information concerning the page referencing behavior of a job, along with how it relates to CPU usage would be needed. This is one possible avenue for future research.

Finally, the inclusion of every resource interval in the job script leads to a lengthy representation of a job. Table VII also includes the size of the file generated by each step in the process. Although the script generator reduces the size of the initial event data file, the script file is still sizable (in excess of 36,000 words to represent approximately 500 seconds of job activity). Work is currently under way to find smaller representations using the job script as a starting point. This involves the determination of the "important" characteristics of a job, and the elimination of unneeded information.

TABLE VII.—Cost of Producing Job Scripts

Job Step	Time[1]	I/O Units[2]	Cost[3]	Size of File Created[4]
Software Monitor	13.019	3660	$1.22	112,501
Data Analyzer	101.952	5788	$5.22	139,756
Script Generator	35.168	2255	$1.84	36,480
Script Analysis	8.019	648	$0.54	-------

Observation interval covered 449.371 seconds
Scripts for 16 jobs were produced

[1]CPU seconds used (including time to load the program)
[2]one I/O unit = 1000 characters of data transferred
[3]as computed by the Purdue University Computing Center
[4]number of 60-bit words

CONCLUDING REMARKS

A job script, defined in this paper to be a detailed list of a job's demands for system resources, does form an accurate, reproducible, and useful workload description at the job level. In turn, collections of scripts can be used to completely describe the workload for a system. The level of detail present in these scripts is both a strength and a weakness of this descriptive form; on the plus side, the detail present means that few assumptions are required when using job scripts (all relevant information is present); on the minus side, the volume of information makes it difficult to form overall conclusions and generalizations from the scripts. In a very real sense, you cannot see the forest for the trees.

At Purdue, we have an ongoing project into the development and validation of computer system models. A critical facet of this project is the formulation of techniques which can be used to describe a system workload. Job scripts have emerged as one such technique. Future work will focus on further reduction in the scripts, perhaps along the lines suggested by the phase-cyclic behavior patterns noted above. The goal would be the automatic generation of job descriptions and then system workload descriptions which are not only accurate, reproducible, and useful, but also compact.

Another line of investigation which could be pursued is the reduction of job scripts so that the resulting information could be used to accurately and succinctly characterize a system workload in a form that would assist computer system managers. For example, such descriptions could be essential factors in upgrading existing systems and procurement of new systems. While using job scripts in this fashion can only be conjectured presently, it can be stated that the scripts in their present form do contain the necessary information. The problems lie in the reductions and final form that would be required.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Steven C. Bruell, Purdue University, for his assistance in obtaining the results given in this paper.

REFERENCES

1. Grenander, U. and R. F. Tsao, "Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals," *Statistical Computer Performance Evaluation,* edited by Walter Freiberger, Academic Press, 1972, pp. 3-24.
2. Denning, P. J. and K. C. Kahn, "A Study of Program Locality and Lifetime Functions," *Proceedings of the Fifth Symposium on Operating Systems Principles,* November 1975, pp. 207-216.
3. Denning, P. J., "On Modeling the Behavior of Programs," *Proceeding of AFIPS Conference 40,* 1972 SJCC, pp. 937-944.
4. Ferrari, D., "Improving Locality by Critical Working Sets," *CACM 17,* 11, November 1974, pp. 614-620.
5. Freiberger, W. F., U. Grenander, and P. D. Sampson, "Patterns in Program References," *IBM Journal of Research and Development 19,* 3, May 1975, pp. 230-243.
6. Madison, A. W. and A. P. Batson, "Characteristics of Program Localities," *CACM 19,* 5, May 1976, pp. 285-294.
7. Brundage, R. E. and A. P. Batson, "Computational Processor Demands of ALGOL-60 Programs," *Proceedings of the Fifth Symposium on Operating Systems Principles,* November 1975, pp. 161-168.
8. Joslin, E. O., "Application Benchmarks: The Key to Meaningful Computer Evaluations," *Proceedings 20th ACM National Conference,* 1965, pp. 27-37.
9. Joslin, E. O. and J. J. Aiken, "The Validity of Basing Computer Selections on Benchmark Results," *Computers and Automation 15,* 1, January 1966, pp. 22-23.
10. DeMeis, W. M., and N. Weizer, "Measurement and Analysis of a Demand Paging Time Sharing System," *Proceedings of ACM National Conference,* 1969, pp. 201-216.
11. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM Systems Journal 8,* 4, 1969, pp. 309-318.
12. Wood, D. C. and E. H. Forman, "Throughput Measurement Using A Synthetic Job Stream," *Proceedings AFIPS Conference 39,* FJCC 1971, pp. 51-56.
13. Schwetman, H. D. and J. C. Browne, "An Experimental Study of Computer System Performance," *Proceedings of ACM National Conference,* August 1976, pp. 693-703.
14. Kernighan, B. W. and P. A. Hamilton, "Synthetically Generated Performance Test Loads for Operating Systems," *Proceedings of 1st Annual SIGME Symposium on Measurement and Evaluation,* February 1973, pp. 121-126.
15. Shreenivasan, K. and A. J. Kleinman, "On the Construction of a Representative Synthetic Workload," *CACM 17,* 3, March 1974, pp. 127-133.
16. Agrawala, A. K. and J. M. Mohr, "A Model for Workload Characterization," *Proceedings of Symposium on the Simulation of Computer Systems,* August 1975, pp. 9-18.
17. Cohen, L. J., "Workload Modeling and Projection," *Computer Performance Evaluation (Proceedings of The European Computing Conference on Computer Performance Evaluation),* September 1976, pp. 437-456.
18. Cheng, D. S., "Trace-Driven System Modeling," *IBM Systems Journal 8,* 4, 1969, pp. 280-289.
19. Sherman, S., *Trace-Driven Modeling Studies of the Performance of Computer Systems,* TSN-30, Ph.D. Dissertation, Computation Center, The University of Texas, Austin, Texas, August 1972.
20. Sherman, S. W., J. C. Browne, and F. Baskett, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," *CACM 15,* 12, December 1972, pp. 1063-1069.
21. Keim, Joseph W. and H. D. Schwetman, "Describing Program Behavior in a Multiprogramming Computer System," *Proceedings of Symposium on Simulation of Computer Systems,* August 1975, pp. 21-29.
22. Ewing, Joel C., "CPUMTR Software Event Probe," Purdue University Computing Center Document LO EVNTPE, December 1974.
23. Schwetman, H. D., "Gathering and Analyzing Data From a Computer System: A Case Study," *Proceedings ACM National Conference,* 1975, pp. 112-117.
24. Schwetman, H. D., "Simulation of Computer Systems Using Automatically Generated Load Descriptions," *Proceedings of Winter Simulation Conference,* January 1974, pp. 699-704.
25. Hatfield, D. J. and J. Gerald, "Program Restructuring for Virtual Memory," *IBM Systems Journal 10,* 1971, pp. 168-192.
26. Kahn, Kevin C., *Program Behavior and Load Independent Performance,* Ph.D. Dissertation, Computer Sciences Department, Purdue University, August 1976.
27. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *CACM 16,* 9, September 1973, pp. 527-531.
28. Schwetman, H. D., "Hybrid Simulation Models of Computer Systems," To appear in *CACM.*

Predicting the workload of a computer system*

by A. K. AGRAWALA and J. M. MOHR
Performance Sciences Inc.
Silver Spring, Maryland

INTRODUCTION

It is a commonly observed phenomenon that the workload on a computer system has a tendency to increase, resulting in poorer service or the aquisition of new, more powerful computers. Since the procurement of a large scale computer is a very lengthy process, it becomes crucial to predict the workload a system is likely to have several years into the future.

Most of today's general purpose third generation computers are used in a multiuser, multiapplication environment. The workload handled by a computer is a composite of the computing needs of all of the users; each putting different resource demands on the systems. Frequently, the load increases as new applications are put on the system, or as users find new ways of making use of the facilities of the system. A commonly used practice is to describe the composite workload in terms of measures such as total CPU hours used, the number of terminal connect hours, or the number of transactions handled. While such measures give an idea as to the load a system handles, they are rarely adequate for prediction purposes.

The major, if not the only, purpose of a computer installation is to meet the computing needs of its user population. A computer system is used for different applications, the nature, volume, and characteristics of which change with time. It is these changes that result in a change in the workload on a computer system. But as the changes in different applications may be different, such changes may not be visible from the composite measures. Therefore, the workload prediction techniques have to take into account each of the components of the workload. In this paper we discuss some techniques which are applicable to the workload prediction problem.

Before we can consider the techniques for predicting the workload of a computer, we need techniques for characterizing the current workload of a system. In the second section we review a workload characterization technique which uses clustering to uncover the natural groupings in the workload. In the third section we discuss the applicability of some of the statistical forecasting techniques to the workload prediction problem. In the fourth section, we discuss the ways in which the cluster description of the system's workload may change and the ways in which such a description may be used to predict future workloads on the system. The fifth section of this paper presents an example of a workload prediction study that was carried out using the techniques discussed in the fourth section. The system described in the fifth section is currently being used for both program development and production work.

* This research was supported in part by the National Aeroanutics and Space Administration, Goddard Space Flight Center, under Grant NASA #NAS 5-24092, and in part by the Environmental Protection Agency under Grant R805478-01-0, to the Department of Computer Science, University of Maryland, College Park, Maryland.

THE CLUSTERING APPROACH TO WORKLOAD MODELLING

If a computer system were to be used to carry out a small number of tasks repeatedly, then it would be a rather straightforward task to create an accurate, validated model of the workload. But, for a general purpose computer system that is shared by a large user population creating an accurate workload model is not easy.

In a multiuser environment each user makes a sequence of computational requests to the system. While the request patterns of one user are likely to be independent of the other users present on the system, the time instants at which the requests are made may depend on the system response characteristics, which in turn, depend upon the actions of all of the other users. No reliable models to date have been formulated for the behavior of a user. One often has to infer the behavior of the user based upon the data collected by the system on logs.[1] In its raw form, this information tends to be so voluminous that it can hardly be used directly as a workload model. Therefore, a simplified model has to be created. The problem of formulating accurate workload models then reduces to the problem of finding natural patterns or structures in the huge amount of data available from a system log.

The problem of finding structures in large volumes of data has often been addressed in the field of pattern recognition.[2,3] As was noted in Reference 4, however, there are some very basic differences between the pattern recognition problem and the workload characterization problem. Primarily, the pattern recognition problem assumes the existence of well defined classes, or groupings, in the observed data. The groupings we are attempting to find in the work-

load are strictly for the purposes of modelling the workload and no a priori groupings can be inferred. Several of the pattern analysis techniques can, however, be used for workload modelling. In particular clustering has been used successfully for creating workload models.[4-9]

In using clustering to model the workload, first we select the degree of aggregation by defining the lowest level of detail that we plan to model as a "workstep." Next we describe a workstep as a point in an appropriately defined multi-dimensional space whose axes represent resource usage levels, or other features of a workstep. Next, a set of worksteps is observed on the system and clustered to uncover their natural groupings.

While the validity of cluster based workload models has been well established in several studies cited above, the groupings are shown to change by changing the feature set.[4,6] This is quite natural because no a priori groupings exist and thus changing the feature set changes the perspective of the problem. This gives the analyst some flexibility in producing good models by choosing appropriate feature sets.

As noted in the first section, a problem with workload prediction arises from the fact that different groups of applications are likely to change in different ways. By using a cluster based workload model we take into account the characteristics of such individual groups. Before discussing how we can use such models in prediction let us discuss some of the statistical forecasting techniques in terms of their relationship to the workload prediction problem.

TYPES OF CHANGES OF THE WORKLOAD

Frequently the statistical forecasting techniques[10] reported in the literature address themselves to the forecasting of a single variable. The workload of a computer system can hardly be considered univariate. However, for the purposes of discussion we are going to assume that a single unit of work can be defined and that we can talk about such things as the load on a system doubling, or varying in some way, and that we can plot the workload vs time on a graph.

The statistical forecasting literature discusses four basic types of changes that may occur and methods by which they may be predicted. Let us now examine these four types of changes and discuss them in terms of the workload prediction problem.

First, there may be a random stochastic variation in the amount of work performed over a period of time. Second, the amount of work may be a monotonically increasing or decreasing function of time. Third, the amount of work performed may show an oscillatory nature, that is there may be some type of periodic behavior. Finally, there may be abrupt discontinuities in the workload. Of course combinations of these may also occur. Let us consider each of these changes briefly.

Random variations

As we noted above, on a system which supports a large user community the workload observed on the system is the composite of the workload generated by individual users. As the independent users may not follow any particular patterns in submitting their processing requests, a sequence of requests from a user may be treated as a stochastic point process.[11] The system workload may, therefore, be obtained as a superposition of the individual point processes and may be modelled as a random point process. The exact nature of the point process depends on the size of the user population as well as the characteristics of the individual users.

In addition to the point process type of characteristics of the workload, the amount of work to be performed in response to each request may also show a random variation. If we take, for instance, the case where the machine is used primarily for program development, we would typically expect to be able to describe the load supplied by an individual by a pair of random variables drawn from two, possibly different, distributions. First, there is the random factor determining how often the user submits a job. This will depend upon, among other things, how long it takes him to fix any bug found in his previous run. There will also be a random factor in the amount of work that the next job will perform, which will depend upon the program being developed and the type of bug that next appears. Therefore, some random variations in the load on the system seem quite reasonable.

Due to the randomness of such variations they can only be characterized by describing their probabilistic properties. Very little can be done with respect to forecasting them except to forecast the parameters of their probabilistic descriptions. The distributions of the loads may be useful in determining peak loading conditions.

Monotonic changes

By a monotomic change in the workload we mean that the amount of work performed by the computer system shows a steady change over time. This change may be observed as a trend and may reflect an increasing or a decreasing workload. In practice, however, the workloads of computer systems have a tendency to increase rather than decrease. The trend shown by such changes may follow any pattern e.g., straight line, quadratic, exponential.

There are several physical justifications for the fact that loads on computer systems tend to increase with time. First many programs that have been in use for a long period of time tend to grow larger and slower as changes are made or new features are added. Second, many programs that have running times proportional to the number of records processed tend to grow longer becasue each subsequent run is required to process a larger number of records. Third, many installations that perform program development or program maintenance seem to enlarge their staffs and thus present a larger processing load to the central computer system.

As the monotonic changes are likely to follow well defined trend lines they lend themselves well to forecasting. Under the assumption that the trends do not change, a prediction of the workload can easily be made by following the trends.

Oscillatory loads

The workload on a system may show a periodic behavior, increasing and decreasing in an oscillatory manner. We may argue that this is the most frequently observed behavior of computer system workloads. While a computer system may run 24 hours a day, the loads go up and down depending on the hour of the day. This variation may be connected with the user work hours and work habits. Similar types of oscillatory behavior with different periods may have a number of causes. Many installations have tasks which are performed daily, weekly, biweekly, monthly, quarterly, or yearly.

Another example of this phenomenon can be observed at a typical university computer center where the period of the oscillation is one semester. At the beginning of the semester, the load is comprised of the research users running relatively large jobs. As the semester continues the load gradually changes until by the end of the semester the load is comprised of a large number of students running small jobs. Also, as more projects are due at the end of a semester the load increases significantly at that time.

As the system workload is a composite, it may contain components with different periodic behaviors. For predicting the workload at some future time we need to identify each oscillatory component and its period. Spectral decomposition techniques may be used for this purpose.

Discontinuities

When one plots a graph of the amount of work submitted versus time one may find points of discontinuity in the curve or its derivative. These discontinuities correspond to sudden changes in the load on the system. Such abrupt changes may occur for any one of several reasons.

A frequently observed cause of sudden changes in the load on a computer system are changes in its normal operating procedures. Examples of such changes in operating procedures would include switching major files from tape to disk, or switching major applications from batch to time sharing. Both of these changes are likely to introduce rather large changes in the system's load. In an environment in which separate systems are used for development and production, the changes in the production machines's load will tend to be very abrupt as applications make the transition from the test phase to the production phase.

If a new application is decided upon and a significant number of new programmers are hired to perform the implementation, then one would expect to see the derivative of the graph of load change abruptly and expect to see the load increase rather quickly there after.

Note that abrupt changes causing the discontinuities are a result of external factors and, therefore, cannot be predicted on the basis of information available from within a system. For the models created on the basis of system measurements such changes have to be treated as random phenomenon. Since these changes may affect the loads drastically they have to be taken into account in any workload prediction. However, since these abrupt changes are due to external factors it may be possible to predict them by examining the external factors. The quantitative impact of these abrupt changes may be less precisely known due to the fact that they will occur in the future and thus cannot be measured.

Composite changes in the load

In the discussion above we observed that all four types of changes are likely to occur in a computer system workload. Examining the causes of such changes we note that they are not mutually exclusive and that the workload changes observed on real systems can only be explained in terms of two or more of these. For workload prediction we have to identify these component change behaviors first.

As the system workload may consist of several segments it is not necessary that all segments of the workload will exhibit similar changes in behavior. In fact, we expect that the change in the behavior of individual components of the workload will be much simpler than changes in the behavior of the overall workload. A way of identifying the segments of the workloads is to use the cluster based characterization.

THE CLUSTER DESCRIPTION OF THE WORKLOAD

In this section we consider the ways in which the cluster description of a computer's workload may possibly change with time and how the different types of changes in the workload will be reflected in its cluster description.

Predicting changes in the cluster description

If one is using the clustering approach to describe the system workload, the problem of predicting a future workload on the system corresponds to the problem of finding the cluster description for what the workload will be at some point in the future. As stated earlier, the purpose of clustering the workstep population is to obtain a homogeneous group of worksteps that behave in the same manner. Since the total system workload would be composed of the sum of the work supplied by each cluster one would expect to find the future system workload to be composed of the sum of the work supplied by each cluster after the appropriate modifications have been made to the cluster. Therefore, to find the predicted load on the system we must examine each of the clusters that are present in the system workload and determine how each of these clusters will change. We then make the appropriate modifications to the cluster descriptions and from these descriptions find the description for the entire system load. This operation is likely to be easier and more accurate than trying to deal with the entire workstep population at one time. These operations do of course require a reather detailed knowledge of the membership of each cluster as well as knowledge of how these worksteps have been varying in the past and the types of variation that

these worksteps are likely to exhibit in the future. Basically, all cluster analysis allows us to do is to model small segments of the load independently and gives us ways of combining these models into a model of the total system workload. The prediction of these changes may be accomplished using some of the techniques described earlier.

Changes in the cluster description of the load

A cluster taken as an abstract entity only has a limited number of properties that may vary. A cluster can be thought of as having a location in an appropriate n-dimensional space and can move or drift in this space. A cluster has members, that is, it is composed of some worksteps from the total workstep population and therefore some things can be said about the distribution of points in the cluster. Finally, a cluster can be used to describe some segment of the total system workload. In addition to the information available about the cluster itself we can analyze the members of the cluster and determine any additional information that we need to know such as their arrival rates or what types of work they performed.

The cluster description of the total system workload is composed of a group of clusters obtained from clustering the workstep population. In addition to the description of the individual clusters as described above we also know the proportion of the total system load attributable to each cluster.

Before discussing the ways in which the cluster description of the workload may change, let us consider what it means for a given cluster to appear in two different cluster runs. If the system is running in a production environment in which the same programs are run repeatedly then two clusters obtained by the analysis of data from two different times are considered to represent the same cluster if the cluster obtained from the second run contains most of the members of the cluster obtained from the first run. In a nonproduction environment, where different programs are run each day, this method of identifying clusters cannot be applied. Here we have to ascertain what type of job constitutes the cluster in question and to find the similar type of cluster in the output of the later cluster run. For example, in a university workload one would expect the amount of resources used by small student programs to grow with time. Therefore, if one found a cluster both from the beginning of the semester and a cluster from the end of the semester which contained most of the jobs from the introductory programming class then one would say the two clusters correspond even though no single job appears in both clusters.

Since a cluster has a position or location in an n-dimensional space its location may change with time. This would occur if most of the worksteps which made up the cluster changed their feature values in approximately the same way. If this were the case, and some of the worksteps did not change in the same manner as the rest of the worksteps then those worksteps would move with respect to the cluster mean. They may in fact move far enough away from the cluster mean so as to be eliminated from the cluster entirely. When the mean of a cluster moves it is quite likely that some points not previously in the cluster would become close enough to the cluster mean to be absorbed into the cluster. Thus, when a cluster drifts it is not unusual to find that some old members leave the cluster and some new members are added to the cluster. This type of change in the cluster description would of course correspond to the case where there was some monotonic change in the amount of system load, that is whether some or all of the worksteps originally contained in this cluster increased in size.

If a cluster contains worksteps which change in different ways over a period of time, the cluster may fall apart. If this happens the worksteps which were in this cluster may drift close enough to other clusters and join them. As a result the percentage of the total system workload attributable to each cluster may change also.

Another type of change in the workload that may change the cluster description of the workload is the introduction of new applications to the system, or changes in the operating mode of existing applications. In many cases the easiest way to handle such predicted changes is to treat the new or changed application as if it were a cluster by itself. This may be necessary since one might have to estimate the resource requirements of the new application as no measued data may be available. Therefore, one might want to handle this cluster in a different manner than the clusters that were obtained by observing the load on the system.

The applications that already exist but are having their operating mode changed, such as being changed from batch to timesharing or from tape to disk, could be treated the same as entirely new applications except that somewhat better estimates may be made for their resource requirements. These worksteps can be handled by removing them from the cluster to which they belonged and treating them as if they were a separate cluster by themselves.

An oscillatory change in the amount of work performed by the system, or a random variation in the amount of work performed by the system, would typically not involve a real change in the description of the clusters themselves, but rather would involve a change in the number of points assigned to each cluster. Therefore, the variation in the load would be primarily attributable to changes in the arrival rates for the worksteps belonging to the various clusters.

It should be noted that the above techniques need only be applied if a substantial portion of the load is expected to change. If only a few small worksteps are expected to change then the effect on the total system load may be extremely small and can be isolated to one or two clusters.

Predicting changes in applications

An application which continues to be processed over a period of time is often used to process transactions or records of some type. While the computer resources that will be required may not be easily predicted, in many cases an accurate prediction can be made of the number of transactions, or the number or records that will be processed as

these numbers are tied to external factors that may be precisely known or easily predicted. For example, the rate of increase of a payroll processing program depends on the rate of increase in the size of the employee records file, which in turn depends on the rate at which the employment in the company is going to change. Changes in employment levels are usually controlled by the management and can be used to project the resource requirements of the payroll program.

In cases such as this, a far more accurate prediction of the future systems load may be made by first forecasting the changes that will occur at the applications level and then mapping these changes into changes at the resource usage level. In some cases this method will be the only way in which reasonable estimates may be made of the future systems workload.

When an abrupt change occurs in the load the changes are almost always attributable to external factors such as changes in budget levels, the introduction of new applications, or other factors that are clearly known in advance. In these cases the changes in the computer load must also be predicted at the applications level and then the expected computer load must be determined based on these predictions.

Clearly the accuracy of the predictions of the applications workload will depend on the accuracy with which the influences of the external factors can be determined.

AN EXAMPLE

In this section we present an example of how the techniques described in the preceding section can be used to predict the load on a real system. One of the difficulties in describing the results of a prediction study is that the accuracy of the prediction may not be determined for several years. Therefore, the importance of this example is not that it purports to be a complete prediction of the load on a given system, which it is not, but rather that it serves to illustrate the techniques that were described in the preceding sections.

The data used in this example was drawn from work done predicting the load for an installation that operates two IBM 370/158 computers. These machines, which are used for program development and the execution of several production systems, process approximately 20,000 job steps per month. This workload prediction study was performed because a need for a replacement for the two 370/158's had been identified and it was necessary to produce a benchmark to aid in the procurement of this replacement system. The new system was to be sized such that it would satisfy the needs of the installation for a five to eight year period. Therefore, a benchmark had to be produced to represent the load five years in the future.

This installation was chosen as the example for this paper because the load shows many of the types of variations discussed above. First, the basic data set that is used by some of the production systems will be moved from tape to disk in the near future. In addition, one of the production

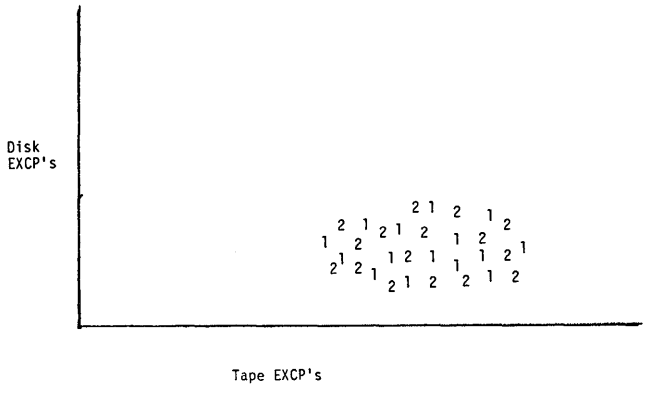


Figure 1a—Present cluster

systems will soon have its operating mode changed from batch to timesharing. The program development portion of the workload can be expected to increase due to the modifications and enhancements being made to the production systems.

Seven features were extracted for each of the job steps and the job steps were clustered using the techniques in References 4-6. This clustering produced twenty clusters, ranging from a cluster of thirty executions of the same workstep to a cluster of over 10,000 relatively small worksteps.

Rather than presenting data on the twenty clusters and over 20,000 data points in a seven dimensional space, the discussion below will deal with only a few of the clusters and some representative data points. In addition, the data presented in the example is presented in terms of only two features rather than the original seven features. We feel that by presenting the data in this manner we will be better able to illustrate the techniques involved than if we had attempted to give a detailed description of all of the data.

The scatter plot of Figure 1a represents a cluster of worksteps from two distinct classes of production systems. For the purposes of this discussion only two features, tape EXCP's and disk EXCP's are shown. The worksteps in the two classes perform essentially the same task, that is editing input data and processing updates to a large master file. There is no way, based upon their resource requirements, that the worksteps from these two classes can be distinguished. Therefore, all of the worksteps are placed into one cluster. The worksteps are described as coming from two distinct classes because the worksteps in class 1 are from a production system that will have its master file moved from tape to disk, while the master file used by the worksteps in class 2 will remain on tape. Therefore, the types of resources required by the worksteps from this class will change rather abruptly. Figure 1b shows a scatter plot of the clusters after predicting the future resource requirements of these worksteps. Since the resource requirements of the worksteps in class 2 are expected to remain essentially constant they are plotted in the same position. The worksteps in class 1 are presented as a new cluster with a significantly lower number of tape EXCP's and a correspondingly higher number of disk EXCP's. These plots are not meant to imply that there

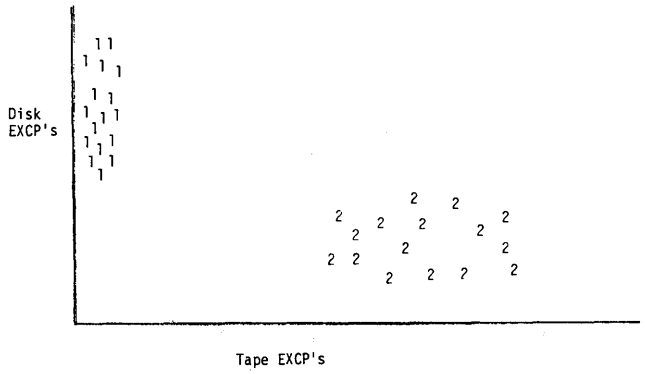


Figure 1b—Predicted clusters after splitting of single cluster

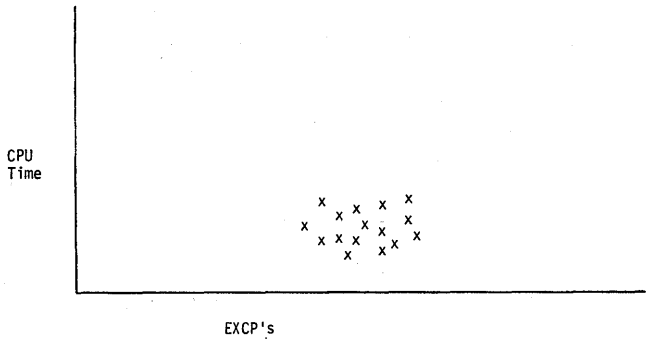


Figure 2a—Present program development cluster

will be a one for one conversion of tape EXCP's to disk EXCP's, but rather that most of the I/O that these worksteps perform will be performed to disk. The exact number of disk EXCP's executed will depend upon a number of factors such as the buffer size that is used, the disk access method that is used, expected changes in the size of the master file, and changes in the number of records to be processed by each workstep. Each of these factors will influence the number of disk EXCP's that a workstep will perform. The number of records to be processed by each workstep may change for several reasons. First, the system will become more heavily used due to increases in the application load. This would increase the amount of I/O performed per day. On the other hand, once the data file is placed on disk, and is thus more accessible, one might decide to run a larger number of worksteps each of which processes a smaller number of records, thereby decreasing the number of disk EXCP's per workstep. Each of these factors depends upon factors to be decided upon before the system conversion can be made and thus can be used in predicting the system load.

The processing load at this installation is subject to a variety of external influences. The processing carried on at this installation is used to support the activities of a large number of divisions throughout the parent organization. Therefore, the processing load on the computer will be effected by management decisions on the initiation or curtailment of activities throughout the organization. In fact, a recent decision by management to reduce a specific activity of one of the divisions will have the effect of reducing the number of records processed by one of the application programs by approximately one third. There is no way that such a change in the processing requirements of the production system could be predicted from data observed on the machine. However, the external factors influencing this change are well understood and their effects can be very accurately determined well in advance. Use of this external information is in fact the only way in which the load on the system due to this application can be accurately predicted.

Figures 2a and 2b represent a cluster that contains many of the program development worksteps. This cluster is composed of a large number of worksteps drawn from a number of sources. Based on observations carried out over a period of time it has been observed that the amount of load attributable to the program development cluster has been increasing. This increase in load is due to an increase in the number and size of the program development worksteps.

This cluster's size and position can be predicted using trend analysis. This is a case in which the position of the cluster drifts due to the movement of most of the data points in the cluster. This movement can be seen when the clusters in Figures 2a and 2b are compared. The cluster in Figure 2b contains more points that the cluster in Figure 2a. This is due to the expected increase in the number of program development worksteps that will be executed by the system.

It should be noted that as the cluster mean drifts and the size of the cluster increases the variance of the data points in the cluster increases as well. This occurs because not all of the worksteps are expected to change in precisely the same manner causing the data points to become more spread out in the seven dimensional space.

When observing this cluster over a long period of time, it becomes clear that the trend is definitely one of constant growth in terms of both the number of worksteps and the size of the worksteps. This is not meant to imply that the rate of increase is absolutely constant. There is some random variation in the number of program development worksteps executed in a given time period. However, this type of random variation is commonly encountered in trend analysis studies and is easily handled.

There is also a rather strong oscillatory component to the workload. While some of the production systems are run on

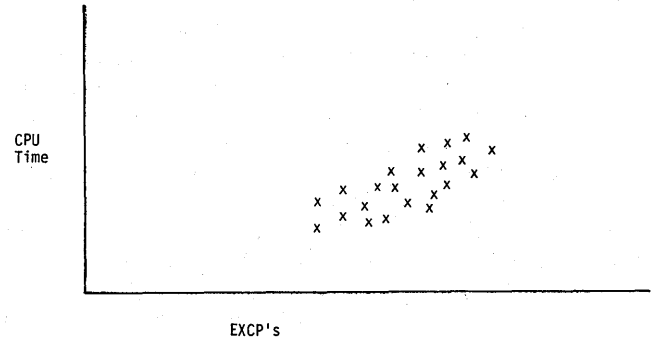


Figure 2b—Predicted program development cluster

TABLE I.—Amount of Load Attributable to Various Clusters

Cluster	Percent of workload normal environment	Percent of workload periodic environment
1	51	10
2	10	2
3	15	1
4	1	84

a daily basis, others are run only once every two weeks. The worksteps from these systems tend to be clustered together due to their rather peculiar I/O requirements. Since the work that these periodic production systems perform is quite time critical they are typically run in stand alone mode so as to insure their timely completion. Therefore, there is a cluster that contributes virtually none of the load most of the time, but every two weeks contributes essentially all of the load. The amount of time required to complete these worksteps is somewhat variable due to such factors as system crashes. Table I contains the percentage of the load contributed by some of the clusters. The two sets of data are for two separate time periods, one which contains the time period in which these periodic systems are running, and the other when the system was running what might be considered its "normal load." The percentage of the load contributed by the non-periodic worksteps is non-zero in the column for the periodic work as the period for which the data was collected overlaps the execution of the periodic systems.

When the set of benchmark programs and the system's requirements are released to the vendors there will be two benchmark experiments to be run. First, a set of benchmark programs representing the normal processing load on the system will be run and the results compared. Second, there will be a requirement that the system to be purchased must be able to execute a representative portion of one of the periodic worksteps in less than a specified amount of time. This has to be done because the system has two operating modes and a new system must be able to handle two very distinct types of loads. It is quite likely that if the worksteps from the periodic systems were combined with the rest of the worksteps into one benchmark then the results that would be obtained from such a benchmark would most likely be rather inaccurate.

CONCLUDING REMARKS

In this paper we consider the problem of workload prediction and note that an accurate prediction can only be made by considering the changes in the individual segments of the workload. For this purpose the segments, or natural groupings, of the workload may be identified using clustering, and then the way components, or clusters, are going to change can be taken into account in the prediction.

The future workload is often affected to a significant degree by external factors. It is crucial that such external factors be taken into account when making the workload predictions by considering the new applications or changes to the modes of operation as well as the effects of external factors on the applications.

REFERENCES

1. Mohr, J. M., A. K. Agrawala, and J. F. Flanagan, "The EXEC-8 Log System, Part I-Description," Univ. of Md., Computer Science Technical Report Series, TR-434 (1976).
2. Kanal, L. N., "Patterns in Pattern Recognition: 1968-1974," *IEEE Trans. on Information Theory,* Vol. IT-20, No. 6, 1974, pp. 697-722.
3. Agrawala, A. K. (ed.), *Machine Recognition of Patterns,* IEEE Press, New York, 1977.
4. Agrawala, A. K., and J. M. Mohr, "The Relationship between the Pattern Recognition Problem and the Workload Characterization Problem," *SIGMETRICS/CMG VIII,* Nov. 29-Dec. 2, 1977, Washington, D.C.
5. Agrawala, A. K., J. M. Mohr, and R. M. Bryant, "An Approach to the Workload Characterization Problem," *Computer,* June 1976, pp. 18-32.
6. Agrawala, A. K., and J. M. Mohr, "Some Results on the Clustering Approach to Workload Modelling," *Proc. CPEUG,* Nor Orleans, Oct. 11-14, 1977.
7. Artis, H. P., "A Technique for Determining the Capacity of a Computer System," *Proc. CPEUG,* Nov. 8-12, 1976, pp. 150-162.
8. Haralambopoulos, G. and G. Nagy, "Data Collection and Analysis of the Distribution of Computing Resources at the University of Nebraska-Lincoln," *Proc. CMG-VII,* Nov. 16-19, 1976, Atlanta, Ga., pp. 144-160.
9. Hunt, E., G. Diehr, and D. Garnatz, "Who are the Users?-An Analysis of Computer Use in a University Computer Center," *SJCC,* Vol. 38, 1971, pp. 231-238.
10. Gilchrist, W., *Statistical Forecasting,* New York, New York, John Wiley and Sons, Inc., 1976.
11. Feller, William, *An Introduction to Probability Theory and its Application, Vol. 2,* New York, New York, John Wiley and Sons, Inc., 1966.

Performance evaluation of nonpreemptive response-ratio schedulers*

by MANFRED RUSCHITZKA
Rutgers University
New Brunswick, New Jersey

INTRODUCTION

The schedulers of nonpreemptive batch processing systems are often based on the first-come first-serve (FCFS) or shortest-job-first (SJF) algorithms. The former is least discriminating in the sense that it does not base its scheduling decisions on the service requirements of jobs. In the SJF scheme, the service requirement of a job (or at least an estimate for it) must be known at the time of arrival. It is used to assure good response for short jobs by discriminating against long ones. Overall, the system response for SJF is better than that for FCFS, but the penalty imposed on long jobs is excessive. Brinch Hansen[1] suggested an alternate scheme which improves the overall response characteristics of FCFS without excessively delaying long jobs. In this scheme, the *highest response-ratio next* (HRN) scheme, the priority of an arriving job is set to zero. It then increases linearly in time at a rate inversely proportional to the service requirement of the particular job. When a job departs, the priorities of all jobs in the queue are evaluated and the one with the highest priority starts to execute.

The expected waiting time (the time between arrival and start of execution, also called the *delay*) of a job conditioned on its service requirement is a common measure for the performance of a scheduling algorithm. We call this conditional expectation the *delay function* $W(x)$, where the parameter x denotes the service requirement. The response time of a job is the sum of its delay in the queue and its service requirement, and the response-ratio is defined as the response time divided by the service requirement. Note that the response-ratio is equal to one plus the delay divided by the service requirement. Overall, the rate at which a job attains service is equal to its service requirement divided by the response time. Thus, its processing rate is equal to the inverse of the response-ratio. By selecting the job with the highest response-ratio for service, the HRN discipline attempts to service competing jobs at equal rates. To quantify the behavior of the HRN scheme, Brinch Hansen derived an approximate expression for its delay function which is valid for a system with Poisson job arrivals and hyperexponentially distributed service times.[1] A comparison of this approximation with the delay functions of FCFS and SJF demonstrates how HRN strikes a balance between these two extremes. In this paper, we derive the exact HRN delay function for M/G/1 systems, i.e., single-server systems with Poisson job arrivals and arbitrarily distributed service times. The derivation amounts to solving an integro-differential equation which was first developed in Reference 1. Making explicit use of random variables, we proceed to formulate this integro-differential equation via a slightly different method.

* The research reported in this paper was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under grant DAHCIS-73-G6.

FORMULATION OF THE INTEGRO-DIFFERENTIAL EQUATION

According to the classification scheme of Ruschitzka and Fabry,[7] a scheduling algorithm is specified in terms of a priority function, a decision mode, and an arbitration rule. The priority function of the HRN scheme is

$$P(r,x)=r/x \tag{1}$$

where the job parameters r and x are the time a job has spent in the system and its service requirement respectively. The decision mode is nonpreemptive, i.e. scheduling decisions are only made when a job departs or when an arriving job finds the system empty. A first-come first-serve arbitration rule will be used to resolve conflicts among jobs with equal highest priority.

Jobs arrive at the M/G/1 system at a rate λ. Their service times S are assumed to be mutually independent and identically distributed. $g(s)$, ES, and ES will denote the probability density function, the expectation, and the second moment of S respectively. The first two moments are assumed to be finite. A job with a service requirement of $S=s$ will be called an *s-job*. $\rho=\lambda ES$ stands for the system load which must be less than one. We express the delay and other random variables of interest for a particular, but arbitrary, job which we call the *test job*. Its service requirement will be denoted by x. For the averages of waiting times we shall use time averages which, due to the assumption of Poisson arrivals, are identical to job averages.

The delay of the test job can be equated with the amount of service performed on other jobs. Using this approach (cf. Reference 6), we distinguish between two groups of jobs: the *early arrivals* which arrive before the test job and the *late arrivals* which arrive after it. Similarly, we define the average *early work* $W_E(x)$ (*late work* $W_L(x)$) as the total amount of service performed on all early (late) arrivals during the test job's lifetime in the system. With these definitions, the average delay of the test job is given by

$$W(x)=W_E(x)+W_L(x). \tag{2}$$

We proceed to further decompose the term for the early work.

When the test job arrives, the system is either empty (idle) or servicing an early arrival. We denote the average remaining service of this early arrival by W_0. More early arrivals may be waiting in the queue. An s-job in the queue with $s\leq x$ will be serviced before the test job since its priority grows faster according to equation (1). The average cumulative service of all these early s-jobs with $s\leq x$ will be denoted by $W_1(x)$. An s-job with $s>x$ will only be serviced before the test job if it has been in the system sufficiently long such that its priority exceeds the faster growing priority of the test job. We call the average cumulative service of these jobs the work $W_2(x)$, i.e., the work associated with those early arrivals that get serviced before the test job although their service requirement is higher. In sum, the average early work encountered by the test job is comprised of three different types:

$$W_E(x)=W_0+W_1(x)+W_2(x). \tag{3}$$

The average time which the test job must wait until the job (if any) which it finds in service completes its remaining service is well-known (c.f. Reference 2, p. 16):

$$W_0=\lambda ES^2/2. \tag{4}$$

In order to obtain expressions for $W_1(x)$ and $W_2(x)$, we first establish an expression for the average number of s-jobs in the queue. Applying Little's theorem[3] to s-jobs in the queue, the average number $n(s)ds$ of s-jobs which the arriving test job finds in the queue is simply

$$n(s)ds=\lambda W(s)g(s)ds. \tag{5}$$

Furthermore, the average cumulative service requirement of all s-jobs in the queue is $sn(s)ds$. Since s-jobs with $s\leq x$ will be serviced before the test job, their contribution to the early work is

$$W_1(x)=\int_0^x sn(s)ds=\lambda\int_0^x sW(s)g(s)ds. \tag{6}$$

Depending on the time it spent in the queue, an s-job with $s>x$ may or may not be serviced prior to the test job. Consider the limiting case in which the priorities of an early arrival and the test job are equal at the instant at which the server begins to work on the *early arrival*. Assuming that the s-job arrived T seconds before the test job and that it has waited V seconds, the test job will have spent $V-T$ seconds in the system. We now equate their priorities according to equation (1) to get

$$V/s=(V-T)/x.$$

Solving for T, we obtain

$$T=V(1-x/s) \tag{7}$$

as the minimum time between the arrivals of the s-job and the test job if the s-job is to be serviced before the test job. Conversely, all s-jobs with $s<x$ that arrive within T seconds prior to the test job's arrival will be serviced after the test job. Due to the assumption of Poisson arrivals, the average number $l(s)ds$ of these latter s-jobs is equal to the rate $\lambda g(s)ds$ of their arrival multiplied by the expected length of T:

$$l(s)ds=\lambda g(s)E[T]ds.$$

After substituting T from equation (7) and using the equality $E[V]=W(s)$ which follows from the definition of the delay function we get

$$l(s)ds=\lambda(1-x/s)W(s)g(s)ds. \tag{8}$$

The average number $n(s)ds$ of s-jobs in the queue is, of course, the sum of the average number $l(s)ds$ of those which get serviced after the test job and some average number $e(s)ds$ of those which are serviced prior to the test job. Thus, we have $e(s)ds=n(s)ds-1(s)ds$, and substituting from equations (5) and (8) yields

$$e(s)ds=\lambda(x/s)W(s)g(s)ds.$$

The average service required by all s-jobs is s times their number, and the average cumulative service of all early arrivals which require $s>x$ seconds of service and get serviced before the test job is therefore

$$W_2(x)=\lambda\int_x^\infty se(s)ds=\lambda x\int_x^\infty W(s)g(s)ds. \tag{9}$$

The early work $W_E(x)$ in equation (3) is now completely determined by equations (4), (6), and (9).

We now proceed to evaluate the late work. Recall that the priority of a late arrival requiring $s>x$ seconds of service grows slower than that of the test job. Such a late arrival will always be serviced after the test job. On the other hand, the priority of a late s-job with $s<x$ may outgrow that of the test job. Assume that a late arrival arrives i seconds after the test job. In order to determine how large the service requirement s of a late arrival may be if it is to be serviced before the test job, we consider the case where the priorities of the late arrival and the test job are equal at the instant at which the *test job's* service commences. Equating their priorities according to equation (1) we get

$$(V-i)/s=V/x$$

where V now denotes the delay of the test job. Solving for s,

$$s=x(1-i/V),$$

yields the maximum service requirement of a job arriving i seconds after the test job if it is to be serviced before the

test job. Holding V constant, we first obtain the average late work conditioned on x and V, i.e., the average cumulative service of all jobs which both arrive and depart during the test job's lifetime in the queue. Making use of Poisson arrivals, the average sum of the individual job's service times may be replaced by an integral over the expected service of a job arriving at time i (time being measured from the arrival of the test job) multiplied by the probability $\lambda\,di$ of its arrival at time i:

$$W_L(x,V)=\int_0^V \int_o^{x(1-i/V)} sg(s)ds\lambda di.$$

Substituting $y=x(1-i/V)$, exchanging the order of integration, and evaluating the inner integral yields

$$W_L(x,V)=\lambda V\int_0^x s(1-s/x)g(s)ds.$$

The average late work can now be obtained by unconditioning this expression with respect to V. With $E[V]=W(x)$, we have

$$W_L(x)=\lambda W(x)\int_0^x s(1-s/x)g(s)ds \tag{10}$$

which is the final expression for the late work.

The average delay $W(x)$ was defined in equation (2). Substituting the early work according to equations (3), (4), (6), and (9) and the late work from equation (10), we get

$$W(x)=W_0+\lambda\int_0^x sW(s)g(s)ds+\lambda x\int_x^\infty W(s)g(s)ds + \lambda W(x)\int_0^x s(1-s/x)g(s)ds.$$

Solving for $W(x)$ yields the final integro-differential equation for the average delay of a job under HRN conditioned on its service requirement x:

$$W(x)=\frac{W_0+\lambda\int_0^x sW(s)g(s)ds+\lambda x\int_x^\infty W(s)g(s)ds}{1-\lambda\int_0^x s(1-s/x)g(s)ds}. \tag{11}$$

This equation describes the HRN delay function in an M/G/1 system. It was first derived by Brinch Hansen[1] in a slightly different manner based on a geometric argument. He then developed an approximate solution for the special case of hyperexponentially distributed service times. We proceed to generalize his results by deriving the exact solution for arbitrary service time distributions.

DERIVATION OF THE DELAY FUNCTION

The integro-differential equation for the delay function $W(x)$ in equation (11) can be reduced to a homogeneous linear differential equation of the second order. We define two auxiliary functions to aid in this reduction process. The first function, $h(x)$, equals the denominator of equation (11).

For $h(x)$ and its derivative we have

$$h(x)=1-\lambda\int_0^x sg(s)ds + (\lambda/x)\int_0^x s^2g(s)ds, \quad h(0)=1, \tag{12}$$

$$h'(x)=-(\lambda/x^2)\int_0^x s^2g(s)ds, \; h'(0)=0. \tag{13}$$

Note for later reference that $h(\infty)=1-\rho$. The second function is defined as $p(x)=xh(x)$. For this auxiliary function and its first two derivatives we have

$$p(x)=x-\lambda x\int_0^x sg(s)ds + \lambda\int_0^x s^2g(s)ds, \qquad p(0)=0, \tag{14}$$

$$p'(x)=1-\lambda\int_0^x sg(s)ds, \qquad p'(0)=1, \tag{15}$$

$$p''(x)=-\lambda xg(x), \qquad p''(0)=0. \tag{16}$$

Note that the functions $h(x)$ and $p(x)$ depend only on λ, x, and g; they are not functions of the HRN discipline. We also define

$$V(x)=W(x)h(x)=W(x)p(x)/x \tag{17}$$

as a more suitable function for the reduction of the integro-differential equation.

With the definitions of $h(x)$ and $V(x)$ in equations (12) and (17), equation (11) becomes

$$V(x)=W_0+\lambda\int_0^x sW(s)g(s)ds+\lambda x\int_x^\infty W(s)g(s)ds,$$

and the first two derivatives of this equation are:

$$V'(x)=\lambda\int_x^\infty W(s)g(s)ds, \tag{18}$$

$$V''(x)=-\lambda W(x)g(x). \tag{19}$$

Using the identities in equations (16) and (17), equation (19) can be rewritten:

$$p(x)V''(x)-p''(x)V(x)=0. \tag{20}$$

Thus, the problem of solving the integro-differential equation (11) for $W(x)$ has been reduced to solving this second-order differential equation for $V(x)$.

We now assume that $V(x)$ has the form

$$V(x)=q(x)p(x) \tag{21}$$

where $q(x)$ is an unknown function to be determined. Substituting equation (21) and its second derivative

$$V''(x)=q''(x)p(x)+2q'(x)p'(x)+q(x)p''(x)$$

in equation (20) yields

$$q''(x)/q'(x)+2p'(x)/p(x)=0.$$

The integral of this equation is

$$\log q'(x)+2\log p(x)=\log C$$

where $\log C$ is an integration constant. Equivalently, $q'(x)=C/p^2(x)$, and a further integration yields the unknown function $q(x)$:

$$q(x)=\int_x^\infty C(1/p^2(y))dy+D \tag{22}$$

where D is a second integration constant. From equations (17) and (21) we have $W(x)=xq(x)$, and with this equality equation (22) becomes

$$W(x)=x\int_x^\infty C(1/p^2(y))dy+Dx. \tag{23}$$

Equation (23) is the desired solution for the HRN delay function, but it remains to determine the integration constants C and D.

Equation (1) states that the priority of a job with a zero service requirement increases at an infinite rate. Such a job will be serviced as soon as the job it finds in service departs. Thus, we have the boundary condition $W(0)=W_0$ which we use to determine the integration constant C. Note that this boundary condition is consistent with our expressions for the early work in equations (3), (4), (6), and (9). Evaluating equation (23) for $x=0$ results in an indeterminate expression (zero times infinity) which can be resolved by L'Hospital's rule

$$W(0)=W_0=C\lim_{x\to 0}\frac{\int_x^\infty (1/p^2(y))dy}{1/x}$$

$$=C\lim_{x\to 0}\frac{-(1/p^2(x))}{-1/x^2}.$$

With $p(x)=xh(x)$ and $h(0)=1$ from equation (12), the value of the integration constant C follows:

$$C=W_0h^2(0)=W_0.$$

The second integration constant, D, may be obtained by using the fact that the average unfinished work U in an M/G/1 system (as seen by a Poisson arrival) is invariant to the scheduling scheme (Reference 2, p. 115):

$$U=W_0/(1-\rho).$$

Expressing U as the average remaining service time of the currently serviced job plus all the work represented by jobs in the queue we have

$$W_0+\int_0^\infty sn(s)ds=W_0/(1-\rho).$$

Substituting the expression for $n(s)ds$ from equation (5) yields

$$W_0+\lambda\int_0^\infty sW(s)g(s)ds=W_0/(1-\rho).$$

We now evaluate this invariance for the HRN discipline by substituting the delay function in equation (23) with $C=W_0$:

$$W_0+\lambda W_0\int_0^\infty s^2\int_s^\infty (1/p^2(y))dyg(s)ds$$

$$+\lambda D\int_0^\infty s^2g(s)ds=W_0/(1-\rho)$$

where the term containing D evaluates as λDES^2 or $2DW_0$. After canceling W_0 and exchanging the order of integration in the double integral, we get

$$1+\lambda\int_0^\infty (1/p^2(y))\int_0^y s^2g(s)dsdy+2D=1/(1-\rho).$$

According to equation (13), the inner integral is equal to $-h'(y)y^2/\lambda$. After evaluating the outer integral by making use of the equalities $p(y)=yh(y)$, $h(0)=1$, and $h(\infty)=1-\rho$ we have

$$1+[1/(1-\rho)-1]+2D=1/(1-\rho).$$

It follows that $D=0$. With the integration constants determined, the delay function in equation (23) becomes

$$W(x)=x\int_x^\infty (W_0/p^2(y))dy. \tag{24}$$

Equivalently, after substituting the auxiliary function p by the defining expression in equation (14) we have

$$W(x)=x\int_x^\infty \frac{W_0}{\left[y-\lambda y\int_0^y sg(s)ds+\lambda\int_0^y s^2g(s)ds\right]^2}dy. \tag{25}$$

The delay functions in equations (24) and (25) are two equivalent forms of the exact solution of the integro-differential equation (11). They specify the average delay of a job, conditioned on its service requirement x, in an M/G/1 system under the HRN discipline.

PROPERTIES OF DELAY FUNCTIONS

The HRN discipline strikes a balance between the extreme scheduling algorithms of FCFS and SJF. It still favors short jobs, but the delay of long jobs is much less excessive than under SJF. Figure 1 illustrates the typical forms of the delay functions of the three scheduling disciplines. The average delay under FCFS is, of course, equal to the unfinished work U:

$$W_{\text{FCFS}}(x)=U=W_0/(1-\rho).$$

It is also independent of the service requirement x. The delay function for SJF was first obtained by Phipps:[4]

$$W_{\text{SJF}}(x)=\frac{W_0}{\left[1-\lambda\int_0^x yg(y)dy\right]^2}.$$

Note that the bracketed term in the denominator is equal to the auxiliary function $p'(x)$ defined in equation (15); it equals one minus the load contribution of all s-jobs with $s<x$. The SJF delay function starts with a value of W_0 and a zero

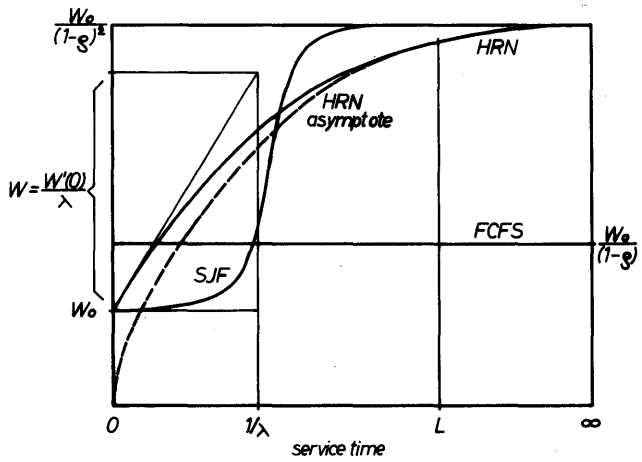


Figure 1—Properties of the delay functions for HRN, SJF, and FCFS

slope at the origin, but rises quite rapidly to its maximum value $W(\infty)=W_0/(1-\rho)^2$. The delay function for HRN covers the same range, but its form is quite different.

The upper integration limits of the two integrals in the definition of $p(x)$ in equation (14) may be replaced by infinity to yield a function which $p(x)$ will asymptotically approach. For very large values of x, say $x \geq L$, we shall use this asymptote as an approximation for the auxiliary function $p(x)$:

$$p(x)=x-\lambda x E S+\lambda E S^2, \qquad x \geq L.$$

The value of L will vary with the given distribution g and the desired accuracy. With this approximation, the delay function in equation (24) can be evaluated analytically,

$$W(x)=W_0/[(1-\rho)(1-\rho+\lambda E S^2/x)], \quad x \geq L, \quad (26)$$

to provide an asymptote for the delay function. This asymptote is depicted in Figure 1. Note that the value of $W(\infty)=W_0/(1-\rho)^2$ can be obtained from this asymptote. The slope of the delay function at the origin may be used to graphically determine the overall average delay W. By definition,

$$W=\int_0^\infty W(s)g(s)ds, \quad (27)$$

and according to equation (18) this expression is equal to $V'(0)/\lambda$. From the definition of $V(x)$ in equation (17) and with $h(0)=1$ and $h'(0)=0$ from equations (12) and (13) we get $V'(0)=W'(0)$ and therefore

$$W=W'(0)/\lambda. \quad (28)$$

Figure 1 illustrates how this equality may be used to obtain the overall average delay W from a plot of the HRN delay function.

So far, we have obtained elementary expressions for $W(0)$, $W(\infty)$, and the asymptote for large service times in equation (26). For intermediate service time values of the delay function, the integral in equation (24) can be evaluated analytically for simple service time distributions only. In general, we will depend on numerical methods for their evaluation. In order to deal with the upper (infinite) integration limit, we split the integration interval into two partitions, $[x,L)$ and $[L,\infty)$. The integral over the latter can then be obtained analytically by making use of the approximation in equation (26). For the integral over $[x,L)$ we use the trapezoidal integration technique with subinterval size d such that $x=md$ and $L=nd$. Thus, equation (24) becomes

$$W(x)=x\sum_{j=m}^{n-1}\left[\frac{W_0}{p^2(jd)}+\frac{W_0}{p^2((j+1)d)}\right]d/2+xW(L)/L \quad (29)$$

where $W(L)$ is given by equation (26). Since $1/p^2$ decreases monotonically, roundoff errors are reduced by summing in decreasing order of j; this corresponds to an integration from L to the left. When many values of the delay function are needed, an iterative method may be used to increase the efficiency of the computation. This iterative method is based on the relation

$$W(md)=W((m+1)d)\frac{m}{m+1}+dm\left[\frac{W_0}{p^2(md)}+\frac{W_0}{p^2((m+1)d)}\right] \quad (30)$$

which follows from equation (29). The initial value $W(L)=W(nd)$ is again given by equation (26), and the iteration proceeds from $m=n-1$ to $m=1$. For $m=0$, we already obtained $W(0)=W_0$ analytically.

The overall average delay W is an important performance measure. Analogous to the delay function, we use a combination of analytical and numerical methods for its evaluation. Starting from equation (27), we again divide the integration interval into two partitions: $[0,L)$ and $[L,\infty)$. The contribution of the integral over the interval $[L,\infty)$ can be approximated analytically by making use of equation (26). Choosing the simple trapezoidal integration method for the interval $[0,L)$, we get

$$W=\sum_{j=0}^{n-1}[W(jd)g(jd)+W((j+1)d)g((j+1)d)]d/2+W(L)G^c(L) \quad (31)$$

where $L=nd$ and G^c is the complementary cumulative distribution function,

$$G^c(x)=1-G(x)=\int_x^\infty g(s)ds. \quad (32)$$

with $G(x)$ denoting the cumulative distribution function of the service time S. Equation (28) shows the relation between the overall average delay W and the slope of the delay function at the origin. Thus, in addition to its importance as a performance measure, the value of W also aids in verifying the accuracy of the numerical integration underlying equation (30).

THE HRN DELAY FUNCTIONS OF SPECIFIC SYSTEMS

Our result for the HRN delay function in equations (24) or (25) is valid for M/G/1 systems. To illustrate the behavior of the HRN discipline for a representative set of specific

distributions, we present four examples with widely varying coefficients of variation C_V, where C_V is defined as

$$C_V=\sqrt{ES^2-(ES)^2}/ES.$$

In the M/D/1 system, service times are constant and $C_V=0$. Erlangian (hypoexponential, Gamma) distributions are characterized by $C_V<1$; we shall deal with a 2-stage distribution or an $M/E_2/1$ system. With the exponential distribution in an M/M/1 system we cover the case $C_V=1$. Finally, the 2-stage hyperexponential distribution of the $M/H_2/1$ system will represent distributions with $C_V>1$.

The program for the evaluation of delay functions and overall delays was written in SIMULA on a PDP-10. We reduced the subinterval size d until we obtained an accuracy of at least 3 digits; $d=1/100$ was sufficient in all cases. For a given distribution and load, the delay function was evaluated at multiples of the subinterval size according to equation (30), and these values were subsequently used to obtain the overall average delay as indicated by equation (31). The SIMULA EXP-function on the PDP-10 has a minimum (most negative) permissible parameter value which is around -89 [EXP$(-89)=2.23\ 10^{-39}$]. We used this constraint to determine the value of L for the various distributions. We shall list these values of L for the Erlangian, exponential, and hyperexponential distributions. For constant service times, the delay function can be obtained analytically, thus avoiding the use of numerical methods.

The M/D/1 system

For constant service times, the distribution function is given by

$$g(x)=\delta(ES)$$

where δ is Dirac's delta function. For this distribution function, the auxiliary function defined in equation (14) evaluates as

$$p(y)=\begin{cases} y, & y<ES; \\ y-\rho y+\rho ES, & y\geq ES. \end{cases}$$

Due to the simple form of $p(y)$, the delay function can be obtained analytically. Evaluating equation (24) for this distribution, we get

$$W(x)=\begin{cases} W_0(1-\rho+\lambda x)/(1-\rho), & x<ES; \\ W_0 x/[(1-\rho)(x-\rho x+\rho ES)], & x\geq ES. \end{cases}$$

Of course, the probability of any job requiring more or less than exactly ES seconds of service is equal to zero. The average delay of a job is therefore

$$W(ES)=W=W_0/(1-\rho).$$

Note that this expression is equal to $W'(0)/\lambda$. As one should expect in an M/D/1 system, the average delay under HRN is equal to the delays under FCFS or SJF.

The $M/E_2/1$ system

The distribution function of the 2-stage Erlangian distribution is given by

$$g(x)=(2\mu)^2 x\exp(-2\mu x)$$

where $\mu=1/ES$. The second moment is $ES^2=3/(2\mu^2)$. For the complementary distribution function we have

$$G^c(x)=1-(1+2\mu x)\exp(-2\mu x).$$

Evaluating equation (14) for this distribution, we get

$$p(y)=y(1-\rho)+3\rho/(2\mu)-\rho y(\mu y+2+3/(2\mu y))\exp(-2\mu y).$$

For the numerical evaluation of the delay function with $ES=1$, we set $L=44$. Figure 2 depicts the HRN delay function together with those of FCFS and SJF for a load of 75 percent. The average overall delays for FCFS, HRN, and SJF are 2.25, 1.74, and 1.37 respectively.

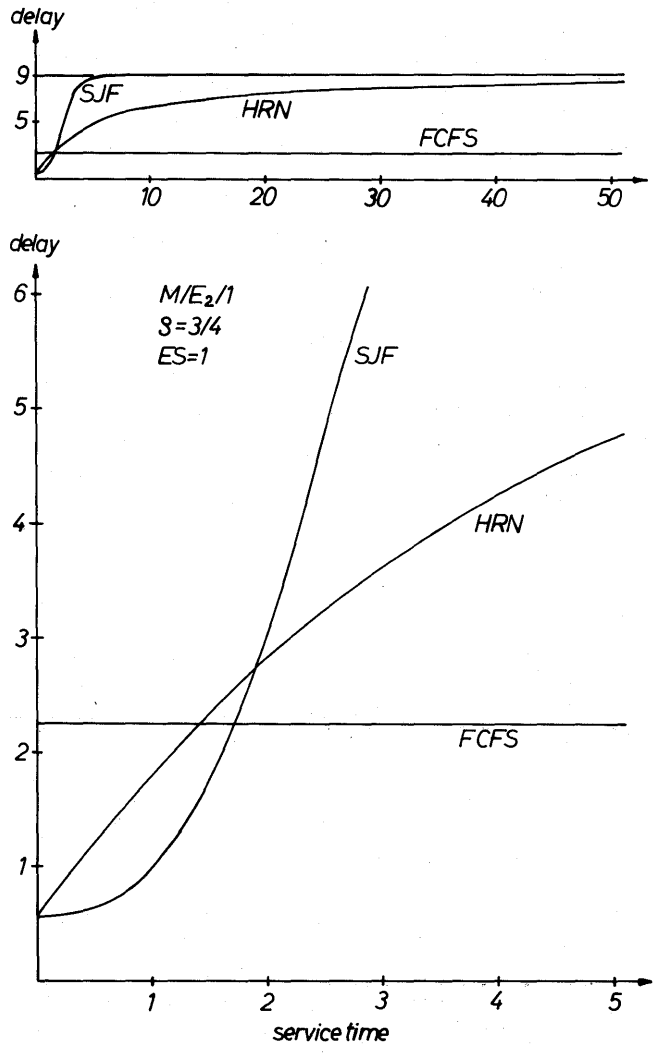


Figure 2—Delay functions for hypoexponentially distributed service times

The M/M/1 system

For the exponential distribution function,

$$g(x)=\mu \exp(-\mu x), \qquad G^c(x)=\exp(-\mu x),$$

the first two moments are given by $ES=1/\mu$ and $ES^2=2/\mu^2$. The auxiliary function in equation (14) specializes to

$$p(y)=y(1-2\rho)+\rho(y+2/\mu)(1-\exp(-\mu y)).$$

Choosing $ES=1$, we set $L=89$ for the evaluation of the delay function based on equation (30). For a load of 75 percent, the delay function of HRN is plotted in Figure 3. For comparison, the delay functions of FCFS and SJF are also illustrated. The average overall delays for FCFS, HRN, and SJF are 3.00, 1.98, and 1.56 respectively.

The M/H$_2$/1 system

The 2-stage hyperexponential distribution function is given by

$$g(x)=ab\mu \exp(-b\mu x)+(1-a)c\mu \exp(-c\mu x)$$

where $\mu=1/ES$, a is an arbitrary probability, and b and c are positive parameters satisfying $a/b+(1-a)/c=1$. The second moment is $ES^2=2[a/(b\mu)^2+(1-a)/(c\mu)^2]$. The complementary distribution function may be obtained from equation (32):

$$G^c(x)=a \exp(-b\mu x)+(1-a) \exp(-c\mu x).$$

The auxiliary function $p(y)$ to be used in the numerical integration specializes to

$$p(y)=y-a\rho y[2-(1+2/(b\mu y))(1-\exp(-b\mu y))]/b$$
$$-(1-a)\rho y[2-(1+2/(c\mu y))(1-\exp(-c\mu y))]/c.$$

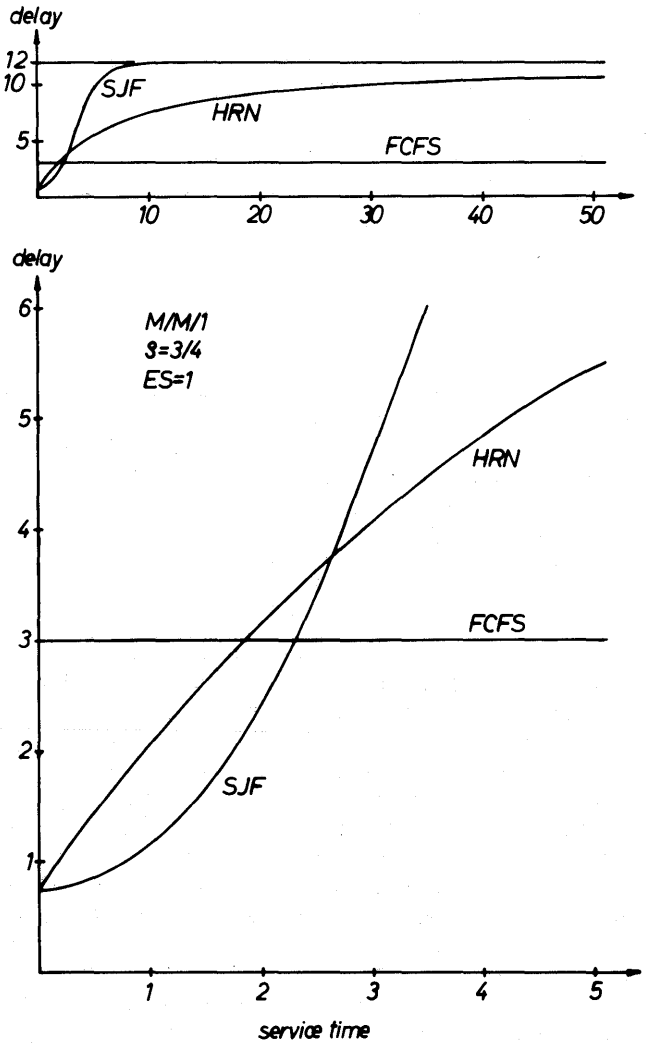


Figure 3—Delay functions for exponentially distributed service times

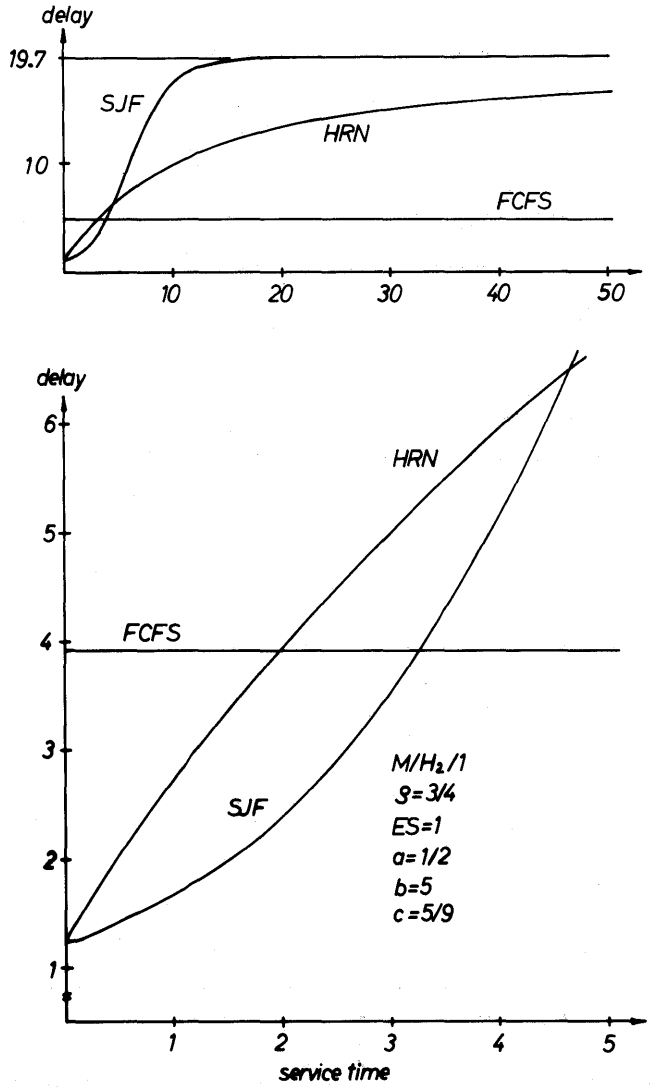


Figure 4—Delay functions for hyperexponentially distributed service times

For the plots in Figure 4 we used the parameters $\mu=1$, $a=\frac{1}{2}$, $b=5$, $c=\frac{5}{9}$, and $\rho=\frac{3}{4}$; L was set to 17. Again, the delay functions of FCFS and SJF are also plotted. The average overall delay of HRN is 2.53 as opposed to 4.92 for FCFS and 2.06 for SJF.

In his paper defining the HRN scheduling scheme,[1] Brinch Hansen also developed an appealingly simple approximation for the delay in an M/H_2/1 system:

$$W(x)=\begin{cases} W_0+x\rho^2/(2(1-\rho)), & x\leq 2W_0/\rho; \\ W_0/[(1-\rho)(1-\rho+2W_0/x)], & x>2W_0/\rho. \end{cases}$$

This approximation was plotted for a load of $\rho=0.71$ and the parameters $a=0.11$, $b=0.21$, and $c=1.88$ which roughly describe a service time distribution which was observed at the University of Michigan.[5] For comparison, we plot the exact solution for this distribution and load in Figure 5 (we set $ES=1$ which determined $L=47$). Again, we also plot the delays for FCFS and SJF. The average overall delay for

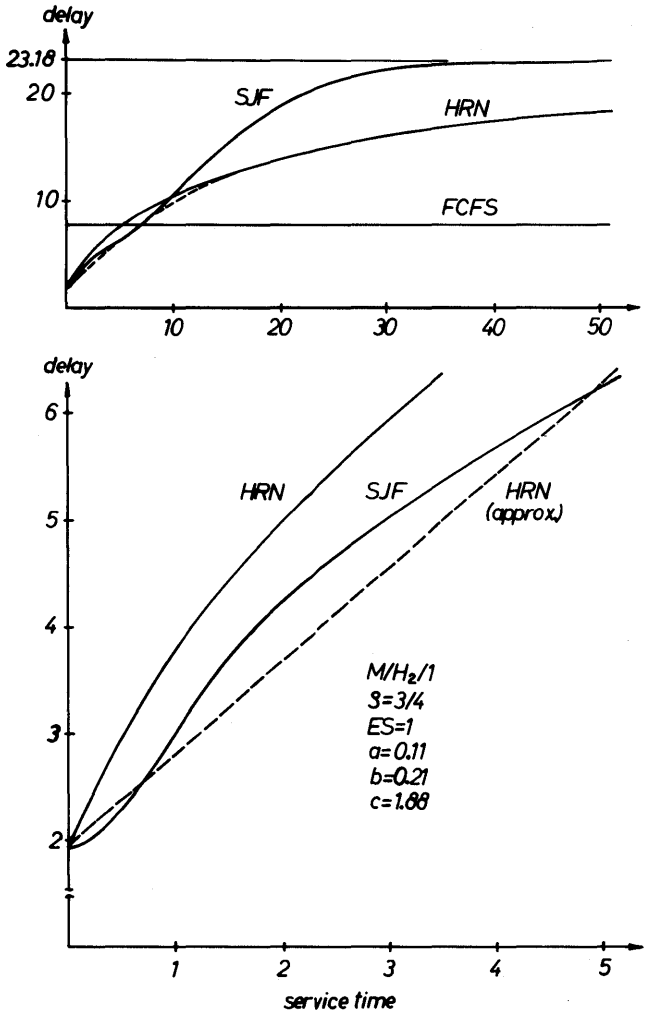


Figure 5—Delay functions for a service time distribution observed at the University of Michigan

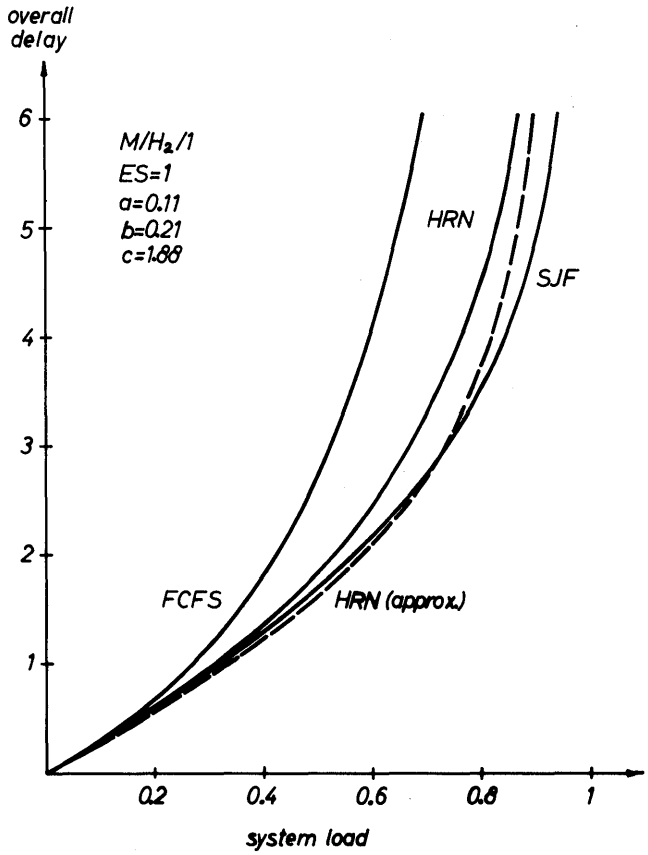


Figure 6—Load-dependency of the overall delay under HRN, SJF, and FCFS

HRN is 3.34 compared to 5.80 for FCFS and 2.84 for SJF. Figure 5 demonstrates the usefulness of the approximate HRN delay function as a rough design tool. Note however, that this function approximates SJF more closely than HRN for service times of up to several times their mean value of one unit of time. This explains why the approximation for the overall average delay W as derived in Reference 1,

$$W=W_0+\rho^2/[2\mu(1-\rho)], \tag{33}$$

evaluates to 2.82, a value which is actually smaller than the overall average delay of 2.84 in the SJF discipline. Figure 6 depicts overall average delays for the University of Michigan distribution as a function of the system load. The broken curve corresponds to equation (33). The solid curves illustrate the load dependencies of the FCFS, the HRN, and the SJF disciplines. While the average overall delay under HRN is necessarily longer than that under SJF, it is still considerably shorter than the overall delay under FCFS. In addition to the preferential treatment of long jobs under HRN as compared to the SJF scheme (which is illustrated by their respective delay functions), the load dependency of the overall delay is a further indication of the superior characteristics of the HRN discipline in a batch-processing environment.

CONCLUSION

The highest response-ratio next (HRN) discipline was first suggested by Brinch Hansen[1] as an attractive alternative to the first-come first-serve (FCFS) and shortest-job-first (SJF) disciplines on which the most common batch-processing algorithms are based. He also formulated an integro-differential equation for the delay function and derived an approximate solution for systems with Poisson job arrivals and hyperexponentially distributed service times.

We redeveloped this integro-differential equation in order to both keep this article self-contained and to present our slightly different derivation technique. We then derived an exact solution for this equation by reducing it to a linear differential equation of the second order. This solution, the delay function for the HRN scheme in equation (25), is valid for Poisson arrivals and arbitrary service time distributions. For general service time distributions, the delay function cannot be expressed in terms of elementary functions. For some simple distributions, however, the integral in the delay function yields to an analytic treatment. As an example of such a simple distribution for which the delay function can be expressed by elementary functions, we presented the case of constant service times. In general, a combination of numerical and analytical methods must be employed to evaluate the delay function. We discussed these hybrid methods and plotted the delay functions for hypoexponentially, exponentially, and hyperexponentially distributed service times. Brinch Hansen's approximation for hyperexponentially distributed service times[1] was compared with our exact result and its accuracy was discussed.

The predominant advantage of the HRN scheme is that it resembles the SJF scheme with respect to the preferential treatment of short jobs without causing the latter's excessive delays for longer jobs. While the average overall delay under HRN is slightly longer than that under SJF, it is still considerably shorter than that under FCFS. Intuitively, these characteristics should be expected as a consequence of the definition of the HRN scheme, but there have been no quantitative measures with the exception of Brinch Hansen's approximate results which are limited to hyperexponential distributions. Our analysis provides the designer with such a quantitative measure which can readily be used to evaluate the performance characteristics of a system under HRN for a given (or anticipated) distribution and load.

ACKNOWLEDGMENT

The author would like to thank his colleagues Bob Vichnevetsky and Jerry Richter for many stimulating discussions concerning the evaluation of delay functions for general service time distributions.

REFERENCES

1. Brinch Hansen, P., "An Analysis of Response Ratio Scheduling," *Proc. IFIP Congress 71*, Ljubljana, August 1971, pp. TA-3:150-154.
2. Kleinrock, L., *Queueing Systems*, Vol. 2: Computer Applications. John Wiley and Sons, New York, 1976.
3. Little, J. D. C., "A Proof for the Queuing Formula: $L=\lambda W$," *Opns. Res.* 9, 1961, pp. 383-387.
4. Phipps, T. E., Jr., "Machine Repair as a Priority Waiting-Line Problem," *Opns. Res.* 4, 1956, pp. 76-85.
5. Rosin, R. F., "Determining a Computing Center Environment," *Comm. ACM* 8, 7, July 1965, pp. 463-468.
6. Ruschitzka, M., "An Analytical Treatment of Policy Function Schedulers," to appear in *Opns. Res.* in early 1978.
7. Ruschitzka, M., and R. S. Fabry, "A Unifying Approach to Scheduling," *Comm. ACM* 20, 7, July 1977, pp. 469-477.

Derivation of equilibrium and time-dependent solutions to M/M/∞//N and M/M/∞ queueing systems using entropy maximization*

by JOHN E. SHORE
Naval Research Laboratory
Washington, DC

INTRODUCTION

Queueing theory has provided the basis for remarkable successes in the performance modeling and analysis of computer systems.[6,19,21] Because it is clear that computer systems do not satisfy assumptions made by the stochastic process models that are used, this success has been somewhat puzzling; it appears that queueing theory equations have wider applicability than is suggested by their classical derivations. Buzen has offered one possible explanation in terms of "operational analysis."[4] In this paper we point out that certain results in elementary queueing theory can be derived simply and with relatively few assumptions by means of entropy maximization. This entropy maximization viewpoint provides the basis of another possible explanation for the widespread applicability of queueing theory formulas.

Analysis by means of entropy maximization has been of interest since Shannon[24] showed, for discrete noiseless systems, that the best encoding of an information source, in the sense of enabling the highest information rate over a fixed capacity channel, is the one that maximizes the source entropy. In addition to continuing applications in communication theory, there has been a growing interest in the use of entropy maximization techniques for probabilistic analysis and problem solving in other fields. Much of this work has been stimulated by that of E. T. Jaynes. Detailed discussions concerning the motivation, justification, and validity of various entropy maximization techniques are available elsewhere.[3,9,12–15,17,26,27] There have been applications in statistical mechanics,[10,11,16] traffic networks,[3,8] reliability estimation,[27] production line decision making,[13,29] system simulation,[5] statistics,[9,20,22] spectral analysis,[1] image reconstruction,[30] and general probabilistic problem solving.[12–15,17,28]

After summarizing the entropy maximization technique in the second section, we obtain in the third section the maximum entropy solution for the state probabilities of an abstract, general system. All of our results are obtained by specializing this general solution. We make some general remarks about applications in the fourth section, and in the fifth section we apply the model to M/M/∞//N and M/M/∞ queueing systems and show how the classical equilibrium distributions result. In the sixth section we discuss how maximum entropy techniques can be used in deriving time-dependent results. After a preliminary application in which we consider a pure birth process, we derive time-dependent distributions for the M/M/∞//N and M/M/∞ queues. Discussion follows in the last section.

ENTROPY MAXIMIZATION

Given a set of independent propositions $\{S_i\}$ that enumerate all of the possibilities in some situation or problem, and given relevant information that is not expressed directly as probabilities, one frequently wishes to convert this information into suitable probability assignments $p(S_i)$ for each of the propositions. Entropy maximization is a means for doing so. In the usual case, known information is expressed as expectations of suitable functions $f_l(S_i)$ defined on the set of propositions. The entropy

$$H=-\sum_i p(S_i)\log p(S_i) \tag{1}$$

is then maximized subject to the constraints

$$\sum_i p(S_i)=1 \tag{2}$$

$$\sum_i f_l(S_i)p(S_i)=\langle f_l\rangle \qquad (l=1,2,\ldots). \tag{3}$$

The well-known solution is

$$p(S_i)=\exp\left[-\beta_0-\sum_l f_l(S_i)\beta_l\right], \tag{4}$$

where β_0 is a Lagrangian multiplier determined by the normalization constraint (2), and where each β_l is a Lagrangian multiplier determined by a known expectation $\langle f_l\rangle$. If the "partition function"

$$Z=\exp(\beta_0)=\sum_i \exp\left[-\sum_l f_l(S_i)\beta_l\right] \tag{5}$$

* The results in this paper were presented orally at the 1976 IEEE International Symposium on Information Theory.

can be evaluated, then the relationship between each Lagrangian multiplier β_l and its associated constraint $\langle f_l \rangle$ is determined by the equation

$$\langle f_l \rangle = -\frac{\partial \beta_0}{\partial \beta_l} = -\frac{1}{Z}\frac{\partial Z}{\partial \beta_l}. \tag{6}$$

For continuous functions, the appropriate generalization appears to be the maximization of "cross-entropy"

$$H = -\int dx\, p(x) \log [p(x)/q(x)] \tag{7}$$

where $q(x)$ is some "prior" distribution. Like (1), (7) originated with Shannon[25] (Shannon's results for continuous channels and for source rates relative to fidelity evaluations are based on expectations of (7)). Maximization of (7) and related functions is discussed by Good,[9] Kullback,[20] and Kashyap.[17] The term cross-entropy is due to Good.[9] Equation (7) has a discrete analog in cases where a prior distribution is known in addition to the expectations (3) (i.e., given no information about the prior, (1) assumes a uniform prior).

As an example, consider the following problem: It is known that a series of events occurs and that the time between successive events can range between 0 and ∞. It is also known that the average time between successive events is $\langle t \rangle$, i.e., that events occur at the average rate $\lambda = 1/\langle t \rangle$. A function $p(t)$ giving the probability density of the time between events is needed. The result of maximizing entropy subject to the constraint $\int t p(t) dt = \langle t \rangle$ is $p(t) = \lambda e^{-\lambda t}$.[5,28] Thus, when a finite mean is specified, the distribution of inter-arrival times that maximizes entropy is exponential. From the maximum-entropy viewpoint, the term "exponential arrivals" describes processes about which one knows only the average arrival rate. If it is further required that the inter-arrival times are independent, then we are assured of a Poisson process. Derivation by maximum entropy of the full exponential family is discussed by Noonan, et al.[22] and, in a somewhat different form, by Kullback.[20]

MAXIMUM ENTROPY SOLUTION FOR A MODEL SYSTEM

In this section, we will obtain the maximum entropy solution for the state probabilities of an abstract system. All of our later results will be obtained by specializing this general solution. The results presented in this section are similar to those in Reference 14. We consider a system composed of N components, each of which can be in one of two "microstates." A system state S_i is defined as an enumeration of which components are in which microstates. Hence, there are 2^N such system states S_i. If one of the two microstates is specified as the "microstate of interest" (it doesn't matter which one), then we may define the function $n(S_i)$ as

$n(S_i) \equiv$ the number of components that are in the microstate of interest, given that the system is in the system state $S_i (0 \leq n(S_i) \leq N)$.

Let $p(S_i)$ be the probability that the system is in the state S_I. We shall now assume that the expectation of $n(S_i)$ is known, and proceed to maximize the entropy of the distribution $p(S_i)$ given this constraint. Specifically, we shall maximize

$$H = -\sum_{i=1}^{2^N} p(S_i) \log p(S_i)$$

subject to the constraints

$$1 = \sum_{i=1}^{2^N} p(S_i)$$

and

$$\langle n \rangle = \sum_{i=1}^{2^N} n(S_i) p(S_i). \tag{8}$$

In this case, the partition function (5) can be evaluated exactly as follows:

$$\begin{aligned} Z = \sum_{i=1}^{2^N} e^{-\beta_1 n(S_i)} &= \sum_{k=0}^{N} g(k) e^{-\beta_1 k} \\ &= \sum_{k=0}^{N} \binom{N}{k} e^{-\beta_1 k} \\ &= (1 + e^{-\beta_1})^N \end{aligned} \tag{9}$$

In (9), β_1 is a Lagrangian multiplier determined by the constraint (8) and $g(k)$ is the number of system states S_i such that $n(S_i) = k$. The relationship between the multiplier β_1 and the constraint $\langle n \rangle$ can be expressed explicitly by applying (6) to (9). The result is

$$e^{-\beta_1} = \frac{\langle n \rangle / N}{1 - \langle n \rangle / N} \tag{10}$$

Using (9) and (10), one can express the maximum entropy solution (4) for $p(S_i)$ as follows:

$$\begin{aligned} p(S_i) &= (1 + e^{-\beta_1})^{-N} e^{-\beta_1 n(S_i)} \\ &= \left(\frac{\langle n \rangle}{N}\right)^{n(S_i)} \left(1 - \frac{\langle n \rangle}{N}\right)^{N - n(S_i)}. \end{aligned} \tag{11}$$

In deriving (11), we began by maximizing entropy subject to the constraint that the expectation of a function $n(S_i)$ is known. Equation (11) expresses the result explicitly in terms of the function $n(S_i)$ and its known expectation $\langle n \rangle$.

In applying the result (11), we shall be particularly interested in the quantity

$p(k) \equiv$ probability that the system is in any state S_i such that $n(S_i) = k$,

which is obtained easily from (11):

$$\begin{aligned} p(k) &= g(k)(1 + e^{-\beta_1})^{-N} e^{-\beta_1 k} \\ &= \binom{N}{k} (1 + e^{-\beta_1})^{-N} e^{-\beta_1 k} \\ &= \binom{N}{k} \left(\frac{\langle n \rangle}{N}\right)^k \left(1 - \frac{\langle n \rangle}{N}\right)^{N-k} \end{aligned} \tag{12}$$

Thus, given the model system and its constraints, the max-

imum entropy distribution for $p(k)$ is a binomial. This derivation of the binomial distribution was first obtained, in a slightly different form, by Jaynes.[14]

In certain applications, one is interested in the limit $N\to\infty$ while $\langle n\rangle$ remains finite. The result for $p(k)$ in this case is obtained by using standard methods (e.g., see Reference 7, p. 142) to derive an approximation to (12) given $\langle n\rangle \ll N$, yielding

$$p(k)=e^{-\langle n\rangle}\frac{\langle n\rangle^k}{k!}. \tag{13}$$

Another probability of interest is

$p_I \equiv$ probability that any given system component is in the "microstate of interest."

Since there are 2^{N-1} system states that meet the requirement that the given component be in the "microstate of interest," p_I is the sum of $p(S_i)$ over these 2^{N-1} states. The function $n(S_i)$ is restricted to the values $n(S_i)=1,2,\ldots,N$. For a particular value $n(S_i)=k$, there are $\binom{N-1}{k-1}$ system states that contribute to the sum of $p(S_i)$. Thus, using (11), we obtain

$$\begin{aligned} p_I &= \sum_{k=1}^{N}\binom{N-1}{k-1}\left(\frac{\langle n\rangle}{N}\right)^k\left(1-\frac{\langle n\rangle}{N}\right)^{N-k} \\ &= \frac{\langle n\rangle}{N}\sum_{l=0}^{N-1}\binom{N-1}{l}\left(\frac{\langle n\rangle}{N}\right)^l\left(1-\frac{\langle n\rangle}{N}\right)^{N-1-l} \\ &= \frac{\langle n\rangle}{N}. \end{aligned} \tag{14}$$

An equivalent form, in terms of the Lagrangian multiplier β_1, is

$$p_I=(1+e^{\beta_1})^{-1}.$$

GENERAL REMARKS ON APPLICATIONS

In general, application of the foregoing results to a given problem requires two steps: First, one must verify that the abstract system on which our results were based is an accurate representation of what is known about the given problem. Specifically, one must make sure that the set $\{S_i\}$ describes completely the set of possibilities implied by the given problem, and that the extent of additional knowledge is limited to the expectation $\langle n\rangle$ of the function $n(S_i)$. If more is known, then our solutions will be incorrect in the sense that additional information is available but not considered, although the solutions may be useful approximations.[14]

The second step is to identify the relationship between $\langle n\rangle$ (or β_1) and parameters contained in the problem statement so that our results for $p(S_i)$ or $p(k)$ can be expressed in terms of these relevant parameters.

As an example, consider the problem of computing the probability of obtaining k heads in N flips of a coin whose single-toss probability of obtaining a head is p_H. In applying out model system, we identify the jth system component as being in the microstate (heads or tails) that records the outcome of the jth coin toss ($1\leq j\leq N$). Thus, a system state represents a record of the results of each toss and the 2^N-element set $\{S_i\}$ is a complete enumeration of the results obtainable by tossing the coin N times. Since p_H is the same as p_I, the combination of (12) and (14) yields the standard result

$$p(k)=\binom{N}{k}p_H{}^k(1-p_H)^{N-k}.$$

APPLICATION TO THE EQUILIBRIUM SOLUTION OF M/M/∞//N AND M/M/∞ QUEUES

In the notation of queueing theory (Reference 18, p. viii), M/M/∞//N refers to a system of N customers and an "infinite" number of servers (effectively equal to N). Both the interarrival times for each customer and the service times for each server are exponentially distributed. Customers arrive for service at an average rate λ per customer and are served at the average rate μ per server ($1/\mu$ is the average time it takes to complete the service of a customer, after which the customer returns to the state "not being served;" $1/\lambda$ is the average time it then takes for that customer to return to the state "being served"). A server is always available to begin service immediately on any customer who needs it. Recall that the specification of only λ and μ is equivalent, from the maximum entropy viewpoint, to the statement that the interarrival and service time distributions are exponential.

In order to apply our model system, we let each system component represent one of the N customers, each of which can be in the microstate of interest "being served" or in the other microstate "not being served." Thus, the 2^N element set $\{S_i\}$ represents all possible states of the M/M/∞//N queueing system. The function $n(S_i)$ is then the number of customers being served given that the system is in state S_i and the expectation $\langle n\rangle$ may be interpreted as a time average. Conventionally (Reference 18, p. 90), $p(k)$ is defined to be the $t\to\infty$ equilibrium limit of a time dependent probability $p(k,t)$. In this context, the probability p_I is the fraction of time that a given customer spends "being served" in a system that has reached equilibrium. The relationship between $\langle n\rangle$ and the parameters of interest (λ,μ,N) is obtained by noting that the following equality must hold at equilibrium:

$$p_I\mu=(1-p_I)\lambda. \tag{15}$$

Hence, using (14) and (10), we have

$$\frac{\lambda}{\mu}=\frac{\langle n\rangle/N}{1-\langle n\rangle/N}=e^{-\beta_1}. \tag{16}$$

Substitution of (16) into (12) yields the following result for the probability that k customers are being served:

$$p(k)=\binom{N}{k}\left(1+\frac{\lambda}{\mu}\right)^{-N}\left(\frac{\lambda}{\mu}\right)^k. \tag{17}$$

This is the classical answer (Reference 18, pp. 107-108).

For the M/M/∞ system, the situation is the same except that the limit $N\to\infty$ is taken and the arrival rate λ is a total

arrival rate (as opposed to an arrival rate per customer) independent of the number of customers in service. In this case, the equilibrium condition is

$$\mu\langle n\rangle=\lambda, \tag{18}$$

which, when combined with (13), yields the classical M/M/∞ result (Reference 18, p. 101):

$$p(k)=e^{-\lambda/\mu}\frac{(\lambda/\mu)^k}{k!}. \tag{19}$$

Based on work by Beneš,[3] a maximum entropy analysis somewhat similar to the foregoing was made by Ferdinand[8] for the machine servicing problem (M/M/1//N) at equilibrium. In that derivation, however, the relationship between the Lagrangian multiplier and the parameters of interest was obtained by assumption and analogy, rather than by derivation. Specifically, for a system in a state S_i with $n(S_i)=k$, Ferdinand assumed a state "energy" $E(k)=-k\log(\lambda/\mu)$ and was thereby able to eliminate the Lagrangian multiplier by analogy with statistical mechanics. In obtaining (17) and (19), on the other, we eliminated the Lagrangian multiplier by means of (10) and the equilibrium conditions (15) and (18).

TIME DEPENDENT SOLUTIONS

The $\langle n\rangle$-constraint equation (8) can represent knowledge of $\langle n\rangle$ at any particular time; the maximum entropy solutions $p(S_i)$ or $p(k)$ then apply to that time. It was only in the last section that we chose a time ($t\to\infty$) that corresponded, in the queueing systems, to equilibrium values of $\langle n\rangle$. Time-dependent knowledge $\langle n;t\rangle$ of the constraint (8) can therefore be substituted meaningfully into (12) or (13) in order to obtain time-dependent probabilities $p(k,t)$. For example, consider a pure birth process in which customers arrive at a constant, total rate λ from an infinite ($N\to\infty$) customer population. Once having arrived, a customer stays forever. The expected number of customers having arrived therefore grows constantly at the rate λ:

$$\frac{d\langle n\rangle}{dt}=\lambda. \tag{20}$$

By substituting the solution of (20) into Eq. (13), and assuming that no customers have yet arrived at $t=0$, we obtain

$$p(k,t)=e^{-\lambda t}\frac{(\lambda t)^k}{k!},$$

which is the classical result (Reference 18, p. 60).

In the last section, we transformed the general solution (12) for $p(k)$ into specific solutions by relating $\langle n\rangle$ or the Lagrangian multiplier β_1 to the physical constraints λ, μ, and N. We did so by writing down the equilibrium conditions (15) and (18), which we repeat here in more explicit form:

$$\frac{d\langle n\rangle}{dt}=-\mu\langle n\rangle+\lambda(N-\langle n\rangle)=0 \quad \text{(M/M/∞//N)} \tag{21}$$

$$\frac{d\langle n\rangle}{dt}=-\mu\langle n\rangle+\lambda=0 \quad \text{(M/M/∞)} \tag{22}$$

Solving these equations for $d\langle n\rangle/dt\neq 0$ would yield time-dependent functions for the expectation $\langle n;t\rangle$. Both (21) and (22) have the form

$$\frac{d\langle n\rangle}{dt}=\alpha-\varphi\langle n\rangle,$$

which has the general solution

$$\langle n;t\rangle=(\alpha/\varphi)(1-e^{-\varphi t})+\langle n;0\rangle e^{-\varphi t}.$$

For the M/M/∞//N case (21), one has $\alpha=\lambda N$ and $\varphi=\mu+\lambda$ so that

$$\langle n;t\rangle=\frac{\lambda N}{\lambda+\mu}(1-e^{-(\lambda+\mu)t})+\langle n;0\rangle e^{-(\lambda+\mu)t}. \tag{23}$$

By assuming that $\langle n;0\rangle=0$ and substituting (23) into (12), we obtain the following time-dependent solution for the M/M/∞//N queue:

$$p(k,t)=\binom{N}{k}\left(\frac{\lambda}{\lambda+\mu}\right)^k[1-e^{-(\lambda+\mu)t}]^k \times\left[1-\frac{\lambda}{\lambda+\mu}(1-e^{-(\lambda+\mu)t})\right]^{N-k}.$$

This result appears not to be well-known among queueing theorists, although it can be derived from a well-known result in reliability theory (Reference 2, p. 78).

For the M/M/∞ case (22), one has $\alpha=\lambda$ and $\varphi=\mu$, so that

$$\langle n;t\rangle=(\lambda/\mu)(1-e^{-\mu t})+\langle n;0\rangle e^{-\mu t}. \tag{24}$$

By assuming that $\langle n;0\rangle=0$ and substituting Eq. (25) into (13), we obtain the following time-dependent solution for the M/M/∞ queue:

$$p(k,t)=\frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k(1-e^{-\mu t})^k\exp\left[-\frac{\lambda}{\mu}(1-e^{-\mu t})\right].$$

This is the classical solution, which can be derived from a slightly more general result given by Saaty (Reference 23, pp. 99-100).

DISCUSSION

In what sense does the maximum entropy distribution for our model system relate to observations made on corresponding real systems? Jaynes[14] has shown that the maximum entropy distribution is related to observed frequencies in the following sense: Given the imposed constraints, the maximum entropy distribution can be realized experimentally in overwhelmingly more ways than any other distribution. Thus, if the analysis includes all of the experimentally operative constraints, then the maximum entropy distribution is overwhelmingly the most likely distribution to be observed experimentally.

In general, one can consider experimental confirmation of a maximum entropy distribution as evidence that the operative physical constraints have been accounted for properly in the maximum entropy analysis. Conversely, major discrepancies between an experimentally observed distribution

and a corresponding maximum entropy distribution is evidence that important physical constraints have been overlooked.[14] When a maximum entropy analysis does not include all of the operative physical constraints, the maximum entropy distribution may still be a useful approximation. The accuracy of the approximation will depend on the importance of the ignored constraints.

In the case of the queueing formulas derived in this paper, we should expect them to apply when a real system's states correspond to those of the model ($\{S_i\}$) and when the system's dynamics are most strongly dependent on the constrained mean value $\langle n \rangle$ determined by the stated equilibrium conditions. If these criteria are satisfied, then the system need not satisfy other assumptions that may be made in classical derivations of the queueing formulas. The maximum entropy viewpoint may therefore explain the surprisingly widespread applicability of queueing theory results.

ACKNOWLEDGMENTS

I thank W. S. Ament, E. Freeman, and D. Kaplan for helpful discussions.

REFERENCES

1. Ables, J. G., "Maximum Entropy Spectral Analysis," *Astron. Astrophys. Suppl.*, 15, 1974, pp. 383-393.
2. Barlow, R. E. and F. Proschan, *Mathematical Theory of Reliability*, John Wiley, New York, 1975.
3. Benes, V. E., *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.
4. Buzen, J. P., "Operational Analysis: The Key to the New Generation of Performance Prediction Tools," *Proceedings COMPCON 76*, IEEE Computer Society, Washington, D.C., Sept. 1976, pp. 166-171.
5. Chan, M., "System Simulation and Maximum Entropy," *Operations Research*, 19, 1971, pp. 1751-1753.
6. Chen, P. P., "Queueing Network Model of Interactive Computing Systems," *Proc. IEEE*, 63, June 1975, pp. 954-957.
7. Feller, W., *An Introduction to Probability Theory and Its Applications, Vol. I*, Wiley, New York, 1957.
8. Ferdinand, A. E., "A Statistical Mechanics Approach to Systems Analysis," *IBM J. Res. Develop.*, 1970, pp. 539-547.
9. Good, I. J., "Maximum Entropy for Hypothesis Formulation, Especially for Multidimensional Contingency Tables," *Annals Math. Stat.*, 34, 1963, pp. 911-934.
10. Jaynes, E. T., "Information Theory and Statistical Mechanics I," *Phys. Rev.*, 106, 1957, pp. 620-630.
11. Jaynes, E. T., "Information Theory and Statistical Mechanics II," *Phys. Rev.*, 108, 1957, pp. 171-190.
12. Jaynes, E. T., "Probability Theory in Science and Engineering," Field Research Laboratory, Socony Mobil Oil Company, Inc., Colloquium Lectures in Pure and Applied Science No. 4, 1958.
13. Jaynes, E. T., "New Engineering Applications of Information Theory," *Proceedings of the First Symposium on Engineering Applications of Random Function Theory and Probability*, (Bogdanoff, J. L., Ed.), John Wiley, New York, 1963, pp. 163-203.
14. Jaynes, E. T., "Prior Probabilities," *IEEE Trans. on Systems Science and Cybernetics*, SSC-4, 1968, pp. 227-241.
15. Jaynes, E. T., *Probability Theory in Science and Engineering*, unpublished lecture notes (available from Physics Department, Washington Univ., St. Louis, Mo.), 1972.
16. Katz, A., *Principles of Statistical Mechanics—The Information Theory Approach*, W. H. Freeman Company, New York, 1967.
17. Kashyap, R. L., "Prior Probability and Uncertainty," *IEEE Trans. Infor. Theory*, IT-17, pp. 641-650.
18. Kleinrock, L., *Queueing Systems Vol. I: Theory*, John Wiley, New York, 1975.
19. Kleinrock, L., *Queueing Systems Vol. II: Computer Applications*, John Wiley, New York, 1976.
20. Kullback, S., *Information Theory and Statistics*, Dover, New York, 1968.
21. Muntz, R. R., Analytic Modeling of Interactive Systems, *Proc. IEEE*, 63, June 1975, pp. 946-953.
22. Noonan, J. P., N. S. Tzannes, and T. Costello, "On the Inverse Problem of Entropy Maximizations," *IEEE Trans. Inform. Theory*, IT-22, 1976, pp. 120-123.
23. Saaty, T. L., *Elements of Queueing Theory*, McGraw Hill, New York, 1961.
24. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Tech. Jour.*, 27, 1948, pp. 379-423.
25. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Tech. Jour.*, 27, 1948, pp. 623-656.
26. Shore, J. E., Derivation of Equilibrium and Time-Dependent Solutions to M/M/∞//N and M/M/∞ Queueing Systems Using Entropy Maximization, Naval Research Laboratory Memorandum Report 3233 (1976).
27. Tribus, M., "The Use of the Maximum Entropy Estimate in the Estimation of Reliability," *Recent Developments in Information and Decision Processes*, (Machol, R. E., and Gray, D., Eds.), Macmillan, New York, 1962, pp. 102-139.
28. Tribus, M., *Rational Descriptions, Decisions and Designs*, Pergamon Press, New York, 1969.
29. Tribus, M. and G. Fitts, "The Widget Problem Revisited," *IEEE Trans. on Systems Science and Cybernetics*, SSC-4, 1968, pp. 241-248.
30. Wernecke, S. J., and L. R. D'Addario, "Maximum Entropy Image Reconstruction," *IEEE Trans. on Computers*, C-26, 1977, pp. 351-364.

The cycle time of a class of closed queueing network models

by WE-MIN CHOW
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

Computer system performance is usually evaluated by its degree of congestion. Hence analysis of queueing network plays an important role in this area. Since in a computer system there are two types of participants—processors and jobs—degree of congestion is usually studied from two different aspects accordingly. In addition to system throughput, processor utilization and queue size, system users would particularly be concerned with their own response times. Many fruitful results in network queues have been presented in previous works,[1-4] where queue size distributions were derived. Based upon these distributions, one can easily calculate throughput, utilization and even average response time. However, little knowledge about response time distribution is available. In this paper, we consider a central server type of queueing system and investigate equilibrium behavior of job's cycle time under certain simplified assumptions. Derivations of equations and theorem proofs can be found in References 5-10, and hence are omitted here.

A central server queueing system is a closed network consisting of a single central server and k parallel servers. A job starts his cycle at the time that he enters the queue of the central server (central queue). After having received service from the central server, the job will join one of queues in front of parallel servers (parallel queues) according to a probability distribution. When the job leaves the parallel server, he will re-enter the central queue and at this moment a cycle has just been completed. Considered in the following is a simple central server queueing system serving N identical jobs. Servers are assumed to have independent exponential service time with FIFO queueing discipline. All parallel servers are identical. Without losing generality, it is further assumed that at time $t=0$ a tagged job is about to join the central queue and the system is in equilibrium.

Define

λ=service rate of the central server
μ=service rate of parallel servers
C=cycle time

$$N_i(t)=\begin{cases}\text{number of jobs in the central server at time } t, & i=0,\\ \text{number of jobs in the } i\text{th parallel server at } t, & i=1,\ldots,k.\end{cases}$$

$\bar{N}(t)=(N_0(t),\ldots,N_k(t))$, system state at time t
$p_m{}^i(t;\bar{n})=P[N_i(t)=m \mid \bar{N}(0^+)=\bar{n}]$
$q(\bar{n})=P[\bar{N}(0^+)=\bar{n}]$
$f(t,k)$=cycle time density with k parallel servers.
$\{U_j\}$=exponential random variables with a rate λ
$\{V_j\}$=exponential random variables with a rate μ

$$T(m)=\sum_{j=1}^{m} U_j$$

$$H=\{\bar{n} \mid \sum_{i=0}^{k} n_i=N,\ n_i\geq 0,\ \forall i=0,\ldots,k\}$$

$$H_1=\{\bar{n} \mid \sum_{i=0}^{k} n_i=N,\ n_0\geq 1,\ n_i\geq 0,\ \forall i=1,\ldots,k\}$$

Given that $\bar{N}(0^+)=\bar{n}$ and the system enters state $\bar{m}=(m_0,\ldots,m_k)$ after a random time $T(n_0)$, with probability $1/k$ the tagged job would join the i parallel queue and then his conditional cycle time is defined by

$$C=T(n_0)+\sum_{j=1}^{m_i} V_j \tag{1}$$

In order to find the distribution of C, it suffices to evaluate $q(\bar{n})$ and $P_m{}^i(t;\bar{n})=P[N_i(t)=m \mid \bar{N}(0^+)=\bar{n}]$.

QUEUE SIZE DISTRIBUTION

By using Gordon and Newell's formula,[2]

$$P[\bar{N}(0^+)=\bar{n}]\propto\left(\frac{1}{\lambda}\right)^{n_0}\prod_{j=1}^{k}\left(\frac{1}{k\mu}\right)^{n_j},\ \forall \bar{n}\epsilon H$$

where $\propto$ is read as "is proportional to" it then can be shown[9]

that

$$q(\bar{n}) \propto \left(\frac{1}{\lambda}\right)^{n_0-1} \prod_{j=1}^{k} \left(\frac{1}{k\mu}\right)^{n_j}$$

or

$$q(\bar{n}) = G^{-1} \left(\frac{1}{\lambda}\right)^{n_0-1} \left(\frac{1}{k\mu}\right)^{N-n_0}, \quad \forall \bar{n} \in H_1, \tag{2}$$

where

$$G = \sum_{\bar{n} \in H_1} \left(\frac{1}{\lambda}\right)^{n_0-1} \left(\frac{1}{k\mu}\right)^{N-n_0}.$$

Note that $q(\bar{n}) = q(\bar{m})$ if $n_0 = m_0$. The state space H_1 may be reduced to one dimension by considering $N_0(t)$ only. (This will be discussed later).

To evaluate the conditional probability

$$p_m{}^i(t;\bar{n}) = P[N_{i(t)} = m \mid \bar{N}(0^+) = \bar{n}]$$

we look into the arrival process to a parallel server. The following lemma are due to the fact of exponential service distribution.

Lemma 1[8,9]

Given that $T(n) = \sum_{j=1}^{n} U_j = s$, where U's are *iid* exponential random variables, then the conditional joint distribution of partial sums $\{T(m) \mid m = 1, \ldots, n-1\}$ are uniform over $(0,s)$.

The above statement asserts that conditioning on $\{N_0(0^+) = n$ and $T(n) = s\}$, the first $(n-1)$ departures epochs from the central server (hence the arrival epochs to parallel servers) form a uniform order statistics ranged over $(0,s)$. Since each departure randomly selects his own parallel server, one would expect that the arrival epochs as seen by each parallel server would also be a set of uniform order statistics.

Lemma 2[10]

Given that in the central server queueing system

(i) $N_0(0^+) = n_0$,
(ii) $T(n_0) = s$ and
(iii) during the period $(0,s)$, $w_i (< n_0)$ jobs enter the i parallel server,

then these w_i arrival epochs form a uniform order statistics ranged over $(0,s)$.

The probability distribution of condition (i) is given by (2). Due to exponential assumption, $T(n)$ has an Erlanigian distribution, i.e.,

$$P[T(n) \le t] = \Gamma(t \mid n,\lambda) = \int_0^t \frac{(\lambda u)^{n-1}}{(n-1)!} e^{-\lambda u} \lambda \, du \tag{3}$$

Furthermore, since parallel servers are identical, condition (iii) is subject to a binomial distribution, i.e.,

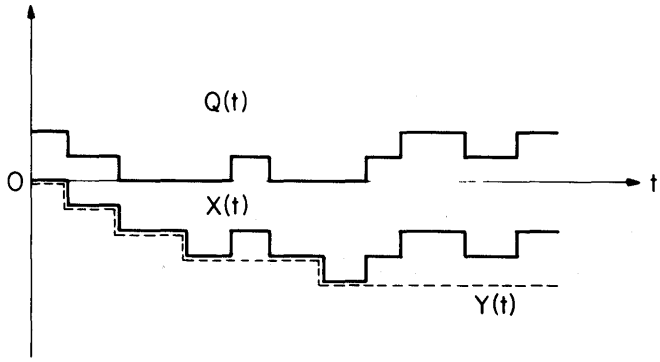


Figure 1—Queue size is characterized by $X(t)$

$$P[W_i(n,s) = w \mid N_0(0^+) = n, T(n) = s] = B(w_i \mid n-1,k) = \binom{n-1}{w_i} \left(\frac{1}{k}\right)^{w_i} \left(1 - \frac{1}{k}\right)^{n-w_i-1}, \quad w_i = 0, 1, \ldots, n-1, \tag{4}$$

where $W_i(n,s)$=number of jobs entered the ith parallel server during $(0,s)$.

By virtue of (2), (3), (4) and Lemma 2, the arrival process to each individual parallel server during the random interval $T(N_0(0^+))$, can be characterized.

Now we investigate transient queueing behavior of parallel servers from which the conditional queue size distribution $P_m{}^i(t;\bar{n})$ is derived.

Consider a single exponential server. Let $A(t)$ be the arrival process and $D(t)$ be the "potential departure process" (departure process when the server is never empty). Hence, $D(t)$ is a counting process such that the interval between successive events is exactly a service time. For a given initial condition the queue size can be characterized by $A(t)$ and $D(t)-$ (see Figure 1).

Lemma 3[5,6,9]

Let $X(t) = A(t) - D(t)$ and $Y(t) = \inf_{0 \le u \le t} X(u)$. Then the queue size $Q(t)$ is given by

$$Q(t) = X(t) + \text{Max}\{Q(0), -Y(t)\} \tag{5}$$

Lemma 4[9]

Let $0 \le S_1 \le \ldots \le S_v \le s$, and $0 \le T_1 \le \ldots \le T_w \le s$ are two independent sets of uniform order statistics over $(0,s)$. Define $Z(t)$ as a one dimension random walk such that (see Figure 2).

$$Z(0) = 0,$$
$$Z(S_i{}^+) = Z(S_i{}^-) - 1,$$
$$Z(T_i{}^+) = Z(T_i{}^-) + 1 \text{ and}$$
$$Z(s) = w - v.$$

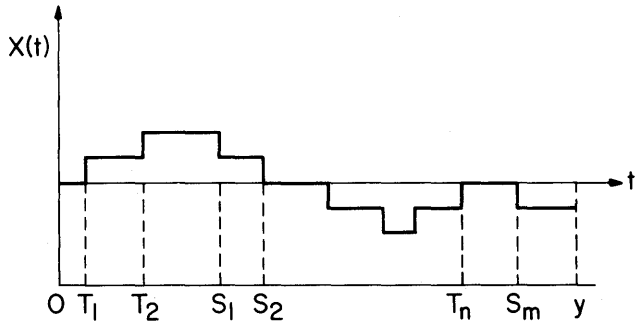


Figure 2—A realization of $X(t)$

Then every realization of the imbedded chain defined by $Z(t)$ at $t=S_i$ or T_i has the same probability $\binom{w+v}{w}^{-1}$.

As far as a parallel server is concerned, $D(t)$ is a Poisson process with a rate μ. It is well-known that the conditional potential departure epochs $\{S_i=\sum_{j=1}^{i} V_j,\ i=1, \ldots, v-1\}$ are uniform order statistics between 0 and s, given that $D(s)=v$. On the other hand, Lemma 2 showed that given (i), (ii) and (iii), the conditional arrival epochs $\{T_1, \ldots, T_w\}$ are also uniform order statistics. In view of Lemmas 3 and 4, we conclude that given $\{N_0(0)=n_0,\ T(n_0)=s,\ A(s)=w,\ D(s)=v\}$, every path defined by $X(t)$ as an imbedded chain for $0\leq t\leq s$ has the same probability.

The reflection principle[7] allows us to verify that the number of paths of a random walk $X(t)$ from (0,0) to (y,u) which cross or touch the line $X(t)=-r$ is equal to the number of paths from $(0,-2r)$ to (y,u) (see Figure 3). Let w (or v) be the number of increments (or decrements) in $X(t)$ by time $t=s$. The total number of possible paths is equal to $\binom{w+v}{w}$ and the number of paths cross or touch the line $X(t)=-r$ is then $\binom{w+v}{w+r}$. Since each path can be realized with an equal chance, the probability

$$P[X(y)\leq -r, \text{ some } y\epsilon(0,s)]=\frac{\binom{w+v}{w+r}}{\binom{w+v}{w}}$$

By definition $Y(t)=\inf_{0\leq y\leq t} X(y)$, it follows that

$$P[Y(s)>-r]=P[X(y)>-r, \text{ for all } y\epsilon(0,s)]$$

$$=1-\frac{\binom{w+v}{w+r}}{\binom{w+v}{w}} \tag{6}$$

By using Lemma 3 and equation (6), it is not difficult to show the following theorem:

Theorem 1[9]

$$P[N_i(s^-)=m_i \mid \bar{N}(0^+)=\bar{n},\ T(n_0)=s,\ W_i(n_0,s)=w_i]$$
$$=r(\mu s, n_i-m_i+w_i)$$
$$-\frac{R(\mu s, n_i+w_i+1)(w_i-m_i-\mu s)w_i!}{(\mu s)^{m_i+1}(w_i-m_i)!}$$
$$=A(m_i \mid n_i, w_i, s, \mu)$$

where

$$r(u,n)=e^{-u}\frac{u^n}{n!}$$

and

$$R(u,n)=\sum_{v=n}^{\infty} e^{-u}\frac{u^v}{v!}.$$

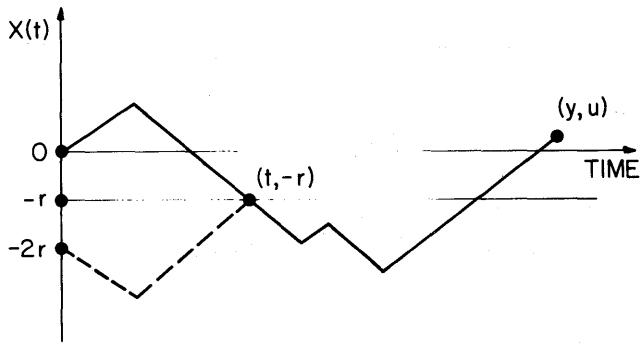


Figure 3—Reflection of $X(t)$

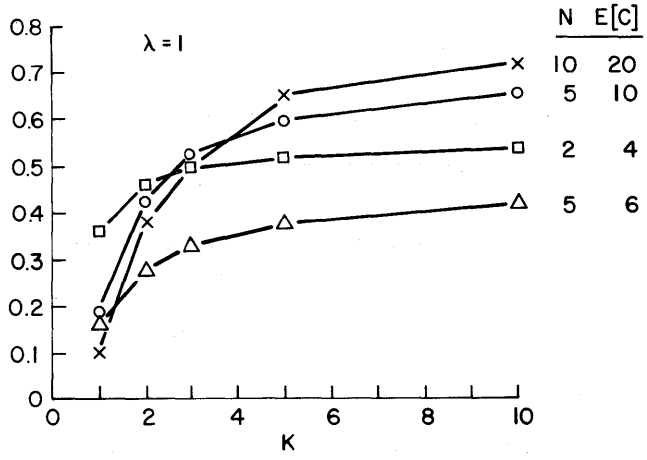


Figure 4—VAR[C]/($E[C]$)^2 when $\lambda=1$

CYCLE TIME

According to equation (1), the conditional cycle time distribution is given by

$$P[C \le t \mid \bar{N}(0)=n,\ T(n_0)=s,\ W(n_0,s)=\bar{w}]$$

$$=\sum_{i=1}^{k}\sum_{m_i=0}^{n_i+w_i}\Gamma(t-s \mid m_i,\mu)\cdot A(m_i, \mid n_i w_i,s,\mu)\cdot\frac{1}{k} \quad (7)$$

Note that since parallel servers are identical, this equation can be simplified. By symmetry the first summation and the factor $1/k$ can be deleted. From this equation, together with (2), (3), (4), and Theorem 1, we can show the following:

Theorem 2[10]

The cycle time density

$$f(t;k)=V^{-1}\sum_{n=0}^{N-1}\sum_{m=0}^{n}\sum_{i=0}^{N-n-1}(k-1)^{n-m}U(k,n,i)$$

$$\left(\frac{\mu}{\lambda}\right)^{n}\Bigg\{r(\mu t,i+m)R(\lambda t,n+1)\binom{n}{m}$$

$$-[r(\mu t,m-1)-r(\mu t,m)]$$

$$[R((\lambda-\mu)t,n-m+1)\lambda^{n+1}(\lambda-\mu)^{m-n-1}\mu^{-m}$$

$$-\sum_{j=0}^{i+m}\left(\frac{\lambda}{\mu}\right)^{m-j}\binom{n+j-m}{j}R(\lambda t,n+j-m+1)]\Bigg\} \quad (8)$$

where

$$V=\frac{1}{\mu}\sum_{j=0}^{N-1}\binom{k+n-i-2}{N-i-1}\left(\frac{\mu}{\lambda}\right)^{i}$$

$$U(k,n,i)=\begin{cases}1, & \text{if } k-1\\ \binom{N-n-i-3+k}{N-n-i-1}, & \text{otherwise}\end{cases}$$

The expected cycle time is

$$E[C]=\frac{N}{\lambda}\left\{1-\frac{\binom{N+k-1}{N}}{\sum_{i=0}^{N}\binom{N+k-1-i}{N-i}\left(\frac{k\mu}{\lambda}\right)^{i}}\right\}^{-1} \quad (9)$$

LIMITING CASES

As mentioned earlier, the state space

$$H_1=[\bar{n} \mid n_0\ge 1,\ n_i\ge 0\ i=1,\ldots,k,\ \sum_{i=0}^{k} n_i=N\}$$

may be reduced to one dimension. Equation (2) implies that

$$P[N_0(0^+)=m]=q(m)$$

$$=\sum_{\bar{n}\ni n_0=m} q(\bar{n}).$$

Hence,

$$q(m)\propto\binom{N-m+k-1}{N-m}\left(\frac{1}{\lambda}\right)^{m-1}\left(\frac{1}{k\mu}\right)^{N-m},\ m=1,2,\ldots,N.$$

Machine Repairman Model, $K=\infty$

If $K=\infty$, it can be shown that

$$q(m)\propto\left(\frac{1}{\lambda}\right)^{m-1}\left(\frac{1}{\mu}\right)^{N-m}\frac{1}{(N-m)!},\ m=1,\ldots,N.$$

and $B(w_i \mid n,\infty)=1$, iff $w_i=0$, for all $i=1,2,\ldots$

The cycle time density becomes

$$f(t;\infty)=\left(\frac{\rho}{\rho-1}\right)^{N}\left(\sum_{i=0}^{N-1}\frac{\rho}{i!}\right)^{-1}$$

$$\mu e^{-\mu t}\left\{\sum_{i=0}^{N-1}\frac{(\rho-1)^i}{i!}[i-(1-\mu t)^i e^{-(\lambda-\mu)t}]\right\} \quad (10)$$

where $\rho=\lambda/\mu$.

Cyclic Queueing Mode, $k=1$

For $k=1$,

$$q(m)\propto\left(\frac{1}{\lambda}\right)^{m-1}\left(\frac{1}{\mu}\right)^{N-m},\ m=1,\ldots N,$$

$$B(w \mid n,1)=1, \text{ iff } w=n-1.$$

The cycle time density becomes

$$f(t;1)=\frac{\mu^N}{\mu^N-\lambda^N}\lambda e^{-\lambda t}\frac{(\lambda t)^{N-1}}{(N-1!)}$$

$$+\frac{\lambda^N}{\lambda^N-\mu^N}\mu e^{-\mu t}\frac{(\mu t)^{N-1}}{(N-1!)} \quad (11)$$

DISCUSSION

In a computer model the central server and parallel servers are usually interpreted as a CPU and k I/O processors, respectively. Therefore, a job's cycle time is equivalent to the interval between his two successive I/O completions. When the number of I/O processors is increased, the system becomes less congested. If the multiprogramming level, the CPU rate and the mean cycle time are fixed, it can be seen from Figure 4 that the ratio of variance to squared mean (V/E)

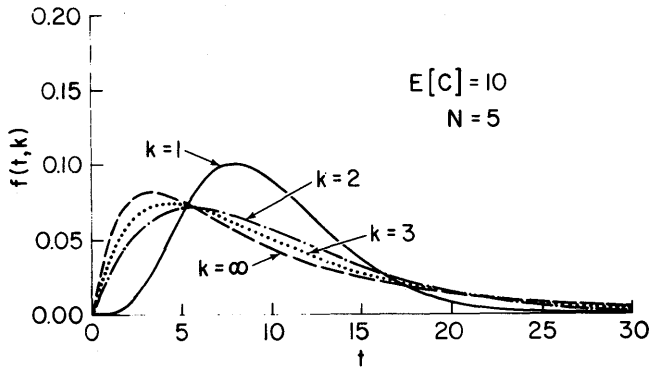


Figure 5—Cycle time densities for different K

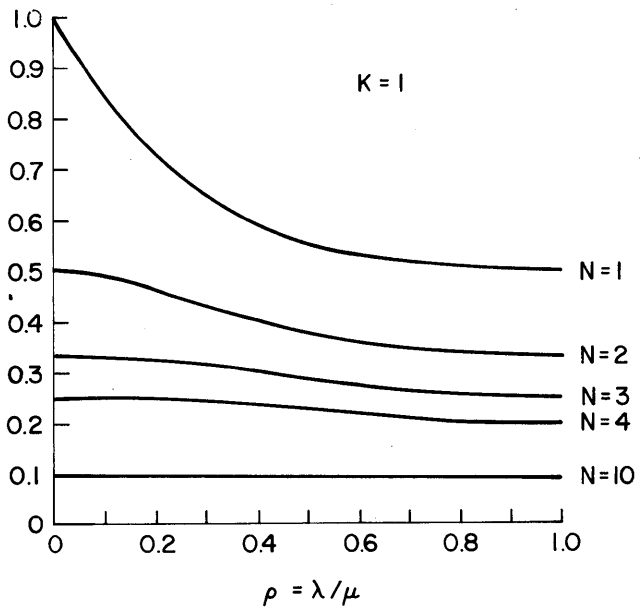


Figure 6—VAR[C]/| $E[C]$ |^2 when k=1 (cyclic queueing model)

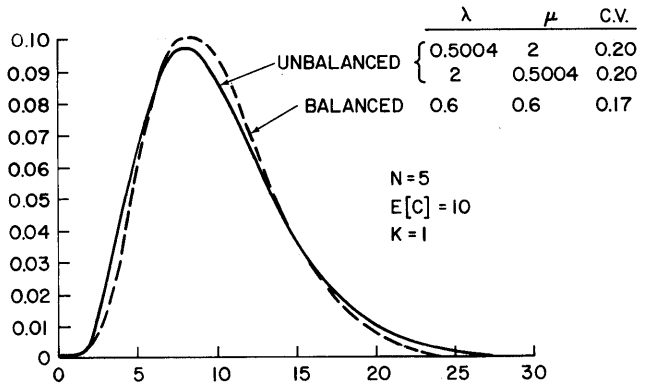


Figure 7—Cycle time densities of cyclic queueing model

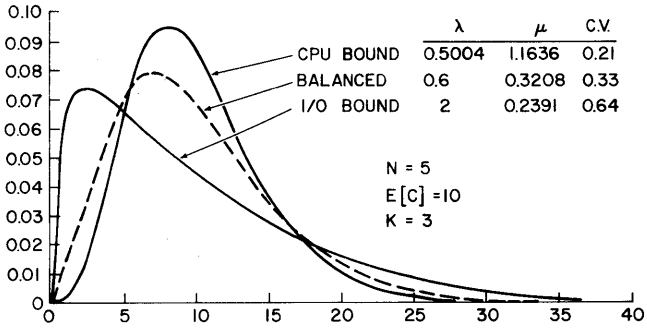


Figure 8—Cycle time densities of central server model

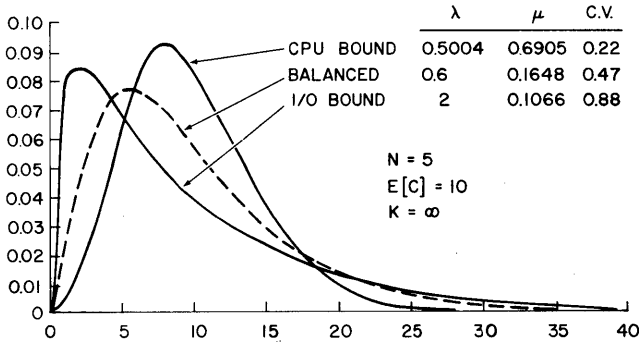


Figure 9—Cycle time densities of machine repairman model

of cycle time is increasing in k. Also, illustrated in Figure 5 is the fact that degree of skewness of cycle time density is reduced if the parameter k is.

When k=1, the cycle time behavior would remain the same if and are interchanged. It can be shown that

$$V/E=\frac{N+1}{N}\cdot\frac{(1-\rho^{N+2})(1-\rho^{N})}{(1-\rho^{N+1})^2}-1, \tag{12}$$

where $\rho=\lambda/\mu$.

This ratio takes its minimum at ρ=1, and is monotonically increasing in ρ for $\rho>1$ (hence decreasing in ρ for $\rho<1$). In other words, the cycle time is more regular in a balance system (i.e., CPU and I/O have compatible processing speeds). However, in a congested system random fluctuation of the cycle time is reduced and its regularity becomes insensitive when ρ changes. As shown by (12), for any positive ρ, $V/E\to 0$ as $N\to\infty$. (See also Figure 6). In fact it is easy to show that

$$f(t;1)=\lambda e^{-\lambda t}\frac{(\lambda t)^N}{N!}, \text{ if } \lambda=\mu \text{ and}$$

$$f(t;1)\to\theta e^{-\theta t}\frac{(\theta t)^{N-1}}{(N-1)!}, \text{ as } N\to\infty$$

where $\theta=\min(\gamma,\mu)$.

TABLE I.—V/E Ratio When N=5 and $E[C]$=10

k		CPU BOUND 0.5004	BALANCED 0.6	I/O BOUND 2
1	μ	2	0.6	0.5004
	V/E	0.20	0.17	0.20
2	μ	1.399	0.393	0.305
	V/E	0.203	0.28	0.51
3	μ	1.164	0.321	0.239
	V/E	0.206	0.33	0.64
5	μ	1.002	0.261	0.186
	V/E	0.21	0.38	0.76
∞	μ	0.691	0.165	0.107
	V/E	0.22	0.47	0.88

For $k>1$, analysis of cycle time is very much complicated. We turn to look at numerical results. Considered are three cases: CPU bound, balanced and I/O bound. For $N=5$ and $E[C]=10$, it is found that V/E ratio decreases as CPU becomes busy. (See Table I. Note that in cyclic queueing model balanced system has the smallest V/E ratio). Figures 7, 8 and 9 present cycle time densities when $k=1$, 3, and ∞, respectively. When CPU is congested, both the machine repairman model and the central server model have less skew densities.

REFERENCES

1. Jackson, J. R., "Jobshop-like Queuing Network," *Management Science*, Vol. 19, No. 1, 1963, pp. 131-142.
2. Gordon, W. J. and G. F. Newell, "Closed Queueing Systems With Exponential Servers," *Operations Research*, Vol. 15, No. 2, 1967, pp. 254-265.
3. Baskett, F., K. M. Chandy, R. R. Muntz and F. Palacios-Gomez, "Open, Closed, and Mixed Network of Queues with Different Classes of Customers," *JACM*, Vol. 22, No. 2, 1975, pp. 248-260.
4. Chandy, K. M., U. Herzog, L. Woo, "Parametric Analysis of Queueing Networks," *IBM Journal of Research and Development*, Vol. 19, No. 1, 1975, pp. 36-42.
5. Champernowne, D. G., "An Elementary Method of Solution of the Queueing Problem With A Single Server And Constant Parameter," *Journal of Royal Statistics Society*, B18, 1956, pp. 125-128.
6. Prabhu, N. U., *Queues And Inventories*, John Wiley, 1965.
7. Feller, W., *An Introduction To Probability Theory And Its Applications*, Vol. 1, 3rd Edition, John Wiley, 1968.
8. Feller, W., *An Introduction To Probability Theory And Its Applications*, Vol. 2, 2nd Edition, John Wiley, 1971.
9. Chow, W., "The Cycle Time Distribution of Exponential Cyclic Queues," RC6484, IBM Research Report, Thomas J. Watson Research Center, Yorktown Heights, NY, 1977.
10. Chow, W., "The Cycle Time Distribution of Exponential Central Server Queues," RC6765, IBM Research Report, Thomas J. Watson Research Center, Yorktown Heights, NY, 1977.

Network access technology—A perspective*

by SHIRLEY WARD WATKINS and STEPHEN R. KIMBLETON
National Bureau of Standards
Washington, DC

INTRODUCTION

The utility of computer networking is widely accepted, as is the complexity of accessing network resources and services. This complexity must be reduced to support more effective user access to and utilization of these resources as discussed in References 1 through 3. Accomplishing this reduction across heterogeneous systems is a primary objective of network access support. Specific capabilities which have currently been shown to be feasible include: "macros" which permit the user to invoke via a single directive a complex of individual system and/or communications subnetwork commands which collectively accomplish a specific and common function such as "login;"[4,5] common command languages for file manipulation and network job execution across a heterogeneous collection of systems;[6] "soft" user interfaces providing command completion, spelling correction, and, potentially, correction of minor syntactic errors;[4,7,8] uniform interfaces to similar services existing on diverse systems such as bibliographic retrieval services;[9,10] and the emergent area of "intelligent assistance."[5]

Currently, there are several ongoing projects providing varying degrees and types of network access support for the interactive user—the population of interest in this paper. Appreciating the functions provided by these efforts and determining additional functions necessary to maximize access support requires establishment of a basic perspective on network access. The global objective of this paper is to provide such a perspective. This will be accomplished through: (i) identifying related research efforts; (ii) identifying problems inhibiting network access support development for the appropriate user categories; and (iii) structuring and discussing the major components of such support. One of the key components of this structure, expert assistance, shields the user from the requirement of learning yet another language—that of the access system. Expert assistance eliminates the requirement that the user become an 'expert' on the access system in order to obtain simplified access to the target system. The sixth section of this paper provides insight into how this goal may be achieved.

Rand Intelligent Terminal Agent (RITA)

The Rand Intelligent Terminal Agent (RITA) has initially been implemented on a shared minicomputer. This implementation is based on production systems (sets of condition-action rules) to encode complex sets of heuristics for handling interactions with users and with external systems.[5,11,12] RITA performs complex time-dependent tasks over extended periods of time with minimal intermediate user input. This is accomplished through user agents (processes operating in behalf of the user). Since the productions systems can modify themselves, RITA agents can learn in the sense of dynamically modifying their behavior.

Two considerations reflected in the design of RITA are that the agents maintain knowledge bases containing heuristic assertions, data reflecting system behavior, and user preferences, and that these knowledge bases must be modifiable by the user. The adaptive behavior of RITA agents represents a sophisticated approach to the area of access assistance.

NBS Network Access Machine (NAM)

The Network Access Machine (NAM) at the National Bureau of Standards is implemented on a PDP 11/45 running the UNIX operating system and represents another shared minicomputer based approach to aiding the network user.[1,3,13,14] Such support is provided through three primary mechanisms: macros which support expansion of simple user-entered commands into the command sequences executable on specific networks and host computers connected to the network; a response analyzer allowing alternative responses (typically the expected response plus error conditions which could occur) during the expansion of a macro; and control mechanisms (case statements, if-then- else, . . .). Collectively, these mechanisms constitute a command language level programming language.

The NAM design is based on the concept of presenting one uniform set of user commands which are executable

* This work is a contribution of the National Bureau of Standards and is not subject to copyright. Partial funding for the preparation of this paper was provided by the U.S. Air Force Rome Air Development Center (RADC) under Contract No. F 30602-77-F-0068. Certain commercial products are identified in this paper in order to adequately specify the procedures being described. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best available for the purpose.

across network boundaries and across heterogeneous host systems. In operation, user commands are first expanded into the command sequence appropriate to the system being accessed, responses are analyzed to determine if the interaction is proceeding as expected, anticipated errors are handled directly, and unanticipated errors are presented to the user for handling. These capabilities have proven sufficiently powerful to support implementation of a prototype common command language for file manipulation and network job execution.[6] In addition, a uniform interface to a collection of bibliographic data retrieval systems has also been demonstrated.[10]

Service specific access support

The use of computer networks has stimulated interest in providing a homogeneous (virtual) interface to multiple heterogeneous resources. One such effort has been pursued by Marcus at MIT in the information services community.[9,15] Access to heterogeneous bibliographic retrieval systems is provided in such a way as to make them appear similar. There are three distinguishing features of this implementation: emphasizing a particular application, providing a uniform virtual interface to heterogenous systems such that all systems appear uniform to the user, and stressing access support to the naive user.

The initial interface, called CONIT (Connector for Networked Information Transfer), is based on the MULTICS system at MIT. Support includes a master index and thesaurus to store the vocabulary of the separate data bases along with index term interrelationships. In addition, the user is provided with a common bibliographic data structure in which data elements are structured and interrelated among different data bases. Once the desired retrieval system is selected, the user enters common commands which are then translated into the specific commands required on the selected retrieval system. The approach has been demonstrated using the ARPANET for access to the National Library of Medicine MEDLINE service and the MIT Intrex retrieval system.

PROBLEMS IN AUTOMATING SERVICE ACCESS

The objective of network usage is access to computer resources or services. Service, here, refers to programs or combinations of programs provided by a computer system: for example, assemblers, compilers, data base management, report generation, and text processing.

Since users access services, the fundamental problem of network access technology is simplification of service access. Moreover, in view of the trend toward interrelated services such as that provided by the National Software Works program production service,[16–18] the problems of interfacing users to services may require consideration of both the interface to a service provided on an individual system as well as to a collection of services provided by several systems. Consequently, Common Command Languages (CCLs) across heterogeneous systems are desirable and have been shown to be feasible.[6]

Simplification of service access requires identification of service user categories and analysis of the factors impeding access simplification.

User categories

Users of a computer system can be divided into three major categories: endusers, applications programmers who construct services for endusers, and systems programmers who provide the environment in which applications programmers function. Note that a given individual may function in all three categories; the importance of the trichotomy is its implicit definition of user objectives.

The global objective of the enduser is access to services. In particular, the nuances of the capabilities provided by a computer communication system, the general characteristics of the system, the locations actually providing the services and the availability of other services not of interest to the user are of secondary interest. The user would prefer that the computer communication system be transparent so that one sees services rather than systems and is only required to provide that information necessary to utilize the service. Further, the user should be relatively unconcerned with the precise structure of system command languages, problems caused by spelling errors and other requirements to conform to the precise syntactic requirements of the system.

Applications programmers construct services. Provided the performance impact is acceptable, high level access support of computer and communication components is very useful. Such support would reduce the need for explicit concern with the vagaries of Job Control Languages, permit common naming conventions across systems, ease access to systems, and support command language level programming.

Systems programmers will often find the software support provided applications programmers of use. However, the detailed, highly system specific knowledge required for the construction of 'systems' software often precludes adoption of a high level view. Although network access support mechanisms can be used by systems programmers, performance constraints will probably limit their applicability. This observation is consistent with the evidence provided by programming language development which demonstrates that for some functions one can only program at the assembly language level.

Looking at the objectives of each user category, it appears that endusers constitute the major user group for network access support. Applications programmers are likely to find such support to be of some help, and systems programmers seem unlikely to be directly aided in the construction or modification of 'system level' functions. Even for this restricted audience, developing the required access support proves difficult.

Access support difficulties

Access support would be comparatively simple if all users had the same problems with the same services. Unfortunately, user perceptions of the difficulty of service access are influenced by:

- user skill
- rate of change
- usage intensity
- access complexity

User skill

Developing effective access support would be difficult even if user skills were uniform across services. However, skills tend to be service dependent and, for each user, span the spectrum from "novice" to "expert." We hypothesize that the desired access support level is inversely correlated with the user's skill and, as a result, there is no single "best" access support methodology.

Rate of change

Rate of change measures the rapidity with which the user changes the collection of accessed services and, thereby, the rate at which the user must learn about the new (for the user) services being accessed. Although we shall not attempt to characterize the precise metric or the difference between 'fast' and 'slow' rates, it is evident that the higher the rate, the greater the learning burden of the user. Thus, we hypothesize that the utility of network access support, as perceived by the user, is positively correlated with the rate of change of service access.

Usage intensity

Usage intensity measures the frequency of use of a given service. We hypothesize that from the viewpoint of an individual user, for services of equivalent complexity (measured in terms of some unspecified metric), the utility of network access support is positively correlated with the (average) time between invocations of the service. This seems intuitively reasonable since the longer the interval between service accesses, the greater the likelihood that required items of information will be forgotten or confused.

Access complexity

Service complexity and capability are often positively correlated. We hypothesize that the desirability of network access support correlates positively with service complexity and, thereby, capability. This seems reasonable since more complex services typically require more complex interactions for their effective utilization.

NETWORK ACCESS OBJECTIVES

Identification of a "best" access support mechanism is a reasonable initial objective for network access investigations. This objective proves unrealistic due to the four factors identified above which, individually, cannot be optimized due to user differences. Heafner found the same difficulty in dealing with user differences for a subset of access support functions, namely command languages. He concluded that there is no "best" command language.[19]

Since developing a "best" approach is infeasible, we consider an alternative—the determination of access functions which are most amenable to automation. Our alternative is based on the identification of: (i) service components appropriate for network access; (ii) support requirements appropriate to these components; and (iii) an integrated set of functions corresponding to these support requirements.

Service access taxonomy

Access to a service proceeds in four stages: acquisition, initialization, utilization and termination. Acquisition encompasses those commands required to 'run' the service together with those required to gain access to the host, if required. Similarly, termination encompasses both the process of "exiting" the service and logging off the host, if appropriate. (It would be inappropriate, or at least inefficient, to log out if the next service to be accessed is also located on the same host.) Initialization covers the setting of parameters prior to actual utilization of the service to accomplish the user's objective.

Given this taxonomy of service access, it follows that the objectives of network access support can be characterized in terms of the capabilities to be provided for each of these stages.

An assertion

We assert that service utilization is not amenable to centralized network level support. Such support can be provided for the other three stages.

Justification of this assertion is provided by both organizational and technological considerations. Organizational justification follows through consideration of the groups which could provide network access support and through analogy with the functions of a computing center counselor.

Network access support of service utilization requires explicit knowledge of the service. Since one would not expect any individual or group of individuals to be knowledgeable in all network accessible services, it follows that no single group can support service utilization. Instead, such support should be provided either by user groups for the

various services, by the service developers (as was done for MYCIN, a diagnostic support service experiment for physicians,[20] and is being done for the ACCAT Command and Control testbed,[7,21,22] or through other means as is being done in the area of providing homogeneous virtual interfaces to heterogeneous bibliographic retrieval services.[9,10,15] In this context, it should be noted that the RITA effort, described earlier is concerned with the development of "smart" user daemons to assist in the utilization of services. (Note that although RITA and MYCIN utilize a similar technology (production systems), RITA support is external to the service while MYCIN provides support as part of the service.)

The technological difficulty in supporting service utilization reflects its implicit requirement of knowledge of both the syntax and semantics of the service. As such, excluding trivial cases, it is effectively a specific instance of the artificial intelligence problem of knowledge representation which, in the general case, is unsolved and still a subject of study.

The following discussion establishes the feasibility of supporting service acquisition, initialization and termination at the network level. Although the difficulties of providing such support for service utilization have been articulated, some discussion of this topic is included for completeness.

Access support objectives

Network access support should simplify access to new (to the user) services and should minimize the need for entering relatively unchanged, previously supplied information when accessing a previously accessed service. We now discuss some of the major requirements implicit in supporting these two objectives.

Accessing new services

Access support of new services requires: (i) identifying the existence of the appropriate service; (ii) providing information about the service; and (iii) simplifying the problems of initial access. Within this paper we shall not consider item (i); however, some consideration has been given to this issue as is reflected in the existence of the ARPANET Resource Handbook[23] and the objectives of the REX system.[24]

Users require two kinds of information to access a service: global descriptive information such as is commonly found in manuals, and specific problem solving oriented information to assist them in more efficient usage of the service. Although either online or hardcopy manuals are appropriate for providing global information, a dynamic mechanism providing highly specific information is required for problem solving. Such information would be based on the individual user and system. Network level support of dynamic assistance seems feasible for the acquisition, initialization and termination stages of service access. Further, it could be provided by the service constructor to support more effective service utilization.

Dynamic service access assistance requires profiles for the acquisition, initialization and termination stages of service access. Such profiles are system dependent and user dependent because access to services is dependent upon the idiosyncracies of the system providing the service and the idiosyncracies of the user accessing the service. Their implementation requires resolution of two issues: (i) determination of the information to be provided, and (ii) development of a mechanism for inserting, storing, modifying and retrieving profiles. For the three stages of interest, handling the service dependency of the profile seems relatively straightforward. However, handling user dependencies can become quite complex if self-tailoring mechanisms are provided.[19]

Accessing previously accessed services

Given that a service has been accessed before, its subsequent access can be simplified through minimizing reentry of previously supplied information via: (i) user and system profiles automating service initialization, and (ii) expert assistance to eliminate the need for explicit user concern with service acquisition, initialization, and termination. Although this reduces the need for entering previously supplied information, it will probably not eliminate it. Consequently, a 'soft user interface' is desirable to accommodate more flexible input than is presently acceptable.

The preceding can be summarized in our assertion that the key network access support components are: (i) profiles for users and services to simplify and automate service acquisition, initialization, utilization and termination; (ii) soft user interfaces to eliminate the need for concern with syntactic 'nits'; (iii) expert assistance to reduce the need for entering repetitive information during the acquisition, initialization and termination stages of accessing previously accessed services; and (iv) dynamic tutorial assistance to provide specific, pinpointed problem solving information online.

The next four sections summarize existing information on the four support components. It should be noted that the soft user interface, expert assistance, and dynamic tutorial assistance require a knowledge base which is provided in profiles. The order in which these components are discussed corresponds to a logical progression providing increasing support to the user (see Figure 1). However, the boundaries between these support functions are not precise and often overlap.

PROFILES

The word "profile" implies an outline or representation of some entity. In access technology a profile is used to represent or characterize a user or service. Several groups have investigated the area of profiles and the information to be maintained therein.[10,19] Since this is an evolving subject area, a succinct list of profile contents is lacking. However, a minimal list of required entries can be suggested.

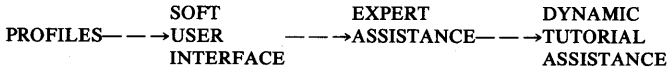


Figure 1—Spectrum for network access support levels

The information in a profile parameterizes both the system and the user for all stages of service access: acquisition, initialization, utilization, and termination. Thus, the profile provides the means to free the user from remembering and executing the mundane, trivial portions of service access. As a specific example, consider the user of multiple systems—each system requiring different connection protocols, passwords, identifications, etc. The profile can be used to store the value of these variables for each system and user, thus relieving the user from remembering these details.

The information stored in profiles is of two different types: static and dynamic. Static refers to entries that are basically unchanging or change very infrequently. The profile entries for acquisition, initialization, and termination contain static entries. Dynamic refers to information which is typified by change. Profile entries associated with service utilization are dynamic; for example, although one may use an information retrieval system daily, it would be expected that key words would not remain the same from session to session. Another example of dynamic profile entries is a function of the access system design rather than the target system. If an objective of the access system is to characterize the user (e.g., experience level) or the use of a system (e.g., certain sequences of commands are habitually repeated), these characterizations are dynamic in nature.

Static profile entries

We know that certain static entries are required for both the user and system profiles. It would appear that such entries are fairly well defined given the access system, target systems, and the communications environment.

User profiles

In the user profile the following entries are required to support the acquisition stage: passwords, user identifications, and account numbers of the user for all known target systems. For the initialization stage, the descriptors required for the target system are any default actions to perform following login to a target system (e.g., any commands which describe special characteristics of a user's terminal) or any default options to issue upon initialization of given services on a target system. If the target system is configured as part of a computer network, the requirement may exist to transfer files from another system to the target system as part of the initialization procedure. Any commands which should be issued prior to finalizing connection with a target system are within the scope of termination. Thus, the user may wish to have certain classes of files deleted prior to logout. If the target system is configured as part of a computer network, the user may wish for all files (or a subset of files) to be transferred to another system in that network prior to disconnection.

System profiles

In the system profile, there are two subcategories of profile entries associated with the acquisition stage: those relative to target systems and those relative to the communications subnetwork (if employed). For the target system, the login sequence is required; for the communications subnetwork, the connection sequence to specify the target system is required along with any required communications control commands.

The profile entries for the initialization stage reflect any requirements of the system prior to utilization. Examples are the syntax required to invoke services, and any identification procedures required prior to accessing given services. Profile entries for the termination stage are in two subcategories: the target system and the communications subnetwork. For the target system, the syntax for logout is required, and for the communications subnetwork, commands for closing a network connection are required.

Dynamic profile entries

While much is known about requirements for static entries in the user and system profiles, determination of appropriate dynamic entries is still an open subject for research. Nevertheless, two major generators of such entries can be identified. Entries corresponding to service utilization are likely to fall into the dynamic category. Further, characterizations of users and modes of access support utilization maintained by the access system require regular updating and, thereby, are likely to be dynamic. We conjecture that dynamic profile entries will be highly dependent upon specific services being accessed and the design of the access system.

SOFT USER INTERFACE

The soft user interface refers to those tasks which humanize the system or network interface to a service for the user. This section structures the functions to be provided by a soft user interface and discusses an approach developed by the Stanford Research Institute.

The functional capabilities provided by a soft user interface can be divided into three major categories: (i) assistance in entering individual commands; (ii) assistance in entering a collection of commands; and (iii) service identification as discussed above. Moreover, each of these categories can be subdivided into two major parts according to whether the actual command syntax is assumed to be fixed. If this syntax is variable, the problem of providing such an interface is closely related to the general problem of restricted natural language interfaces.

Assuming the reader knows the appropriate command,

assistance in its entry can be provided through command completion to minimize the amount of information which must be entered, spelling correction, help features to remind the user of the command syntax, and user oriented diagnostics to facilitate identification of what went wrong.

Assistance in reentering collections of commands can be achieved through allowing the user to catalog command sequences generated in previous interactions with the system. This minimizes the amount of information which must be reentered. An approach to providing such a mechanism is described in the following section.

The preceding has described means for facilitating the entry of commands having rigid syntactic specifications. We now discuss how these restrictions might be reduced through a more sophisticated interface.

Restricted natural language interfaces represent a logical means for providing syntactically and semantically flexible commands. In this context, there is a spectrum of features which can be provided. Thus, the interface may query the user concerning goals and objectives in an attempt to unravel a nebulous user request. Moreover, the interface may use a profile system to facilitate self-tailoring in terms of verbosity and user idiosyncracies. As this query facility is expanded it rapidly approaches the capabilities included in dynamic tutorial assistance discussed below.

As an example of the potential power of a restricted natural language interface, we briefly summarize some of the capabilities of a system called LIFER (Language Interface Facility with Ellipsis and Recursion),[7,8,21] which has been implemented at the Stanford Research Institute. LIFER provides an automatic facility for handling elliptical (incomplete) inputs, a spelling corrector, a grammar editor, and language extension through the use of paraphrase. LIFER is implemented in INTERLISP and consists of two major components. One component is a set of interactive language specification functions which are used to define an application language which is a supset of a natural language. The other component is a parser which interprets natural language inputs to translate them into appropriate interactions with the application software.

Production rules are at the core of the LIFER language specification functions. The production rules are advertised to be easily modified and interactively tested. The intermixing of calls to LIFER for language definition, extension or modification with calls to the parser for utilizing the developing language system serves as an aid for language development. The processing of elliptical inputs is another aid to the language builder. This elliptical handling is provided automatically by LIFER and frees the system builder from incorporating such constructions in the application language.

For spelling correction LIFER makes use of the spelling corrector in INTERLISP. In the event that LIFER does not understand an input it prints "user-oriented" error messages to indicate what it does understand and suggests correctional steps.

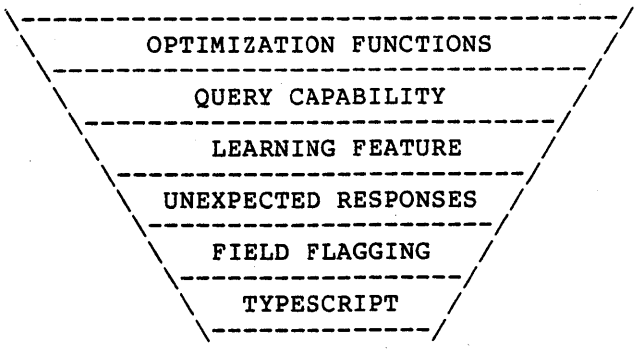


Figure 2—Levels of expert assistance

EXPERT ASSISTANCE

Using an access system often requires that the users learn a new command language—the language of the access system. Generally, system developers provide a menu of existing aids; however, in order for the users to tailor those aids to individual needs, or to create new aids, the users must deal with the language of the access system. This indeed is an ironic consequence since it is the goal of access assistance to lift users above the idiosyncracies of individual systems. Expert assistance refers to the capability of automatic generation of access system commands to achieve the desired user actions.

Levels of expert assistance

There are five increasing levels of sophistication for expert assistance; however, the underlying commonality is that the user informs the expert assistance system of the points at which observation of the user's actions is to begin and end. Figure 2 represents the hierarchy of functions which can be provided in an expert assistance system.

The minimum level of access assistance is provided through a capability to simply record interactions which actually occur between a system and user, and produce the access system command sequences required to replay the recorded interactions. This level of expert assistance is not flexible because it does not handle variations in system response or minor (parameter setting) changes in the user commands. Greater sophistication is desired.

The second level of expert assistance permits identifying certain fields of the user command or system response as variable. Such identification can either be explicitly provided by the user or by the expert assistance system based on its knowledge about commands and services. The identified fields can then be incorporated as parameters in the single command entered by the user to invoke the parameterized sequence of commands entered earlier. As an example, the expert assistance system might recognize the invocation of an editor; given that the syntax for the editor is known, the location of the expected file name would also be known and could be flagged as a parameter.

The major drawback of the second level is the frequent occurrence of unexpected responses common to a networking environment. The remedy is to incorporate handling of

unexpected responses in the assistance system—the third level. Two approaches to their handling can be identified: (i) a priori identification of all possible responses together with incorporation of appropriate actions; and (ii) notifying the user when an unexpected response occurs and permitting the user to specify the next action to be taken.

At the fourth level of sophistication, the expert assistance system draws upon its own knowledge base of services, and uses this information to enhance the generation of command sequences. For example, the expert assistance system may automatically generate appropriate responses to handle system level messages which occur during an interaction and should be ignored rather than generating an error. This level differs from the third level because the expert assistance system dynamically expands its knowledge base rather than having given situations identified at design time and programmed into the system.

At the fifth level of assistance, the user would be able to ask the assistance system to create the specific commands by indicating a general class of activity and a target system.

Currently, expert assistance is viewed as an option available to the user. There is no intent in the overall design to have the expert assistance system suggest to the user that certain activities are being repeated and could be incorporated into a command to the access system; nor is there an intent for the expert assistance system to suggest to the user that there are more efficient ways to achieve the goals being pursued. These capabilities are beyond the current scope of expert assistance. However, there is a sixth possible level, optimization, which may prove desirable as a future goal of such assistance. (Such support mechanisms have been considered for provision as part of a military message processing system.[19])

An implementation approach

The NBS Expert Assistance System (EAS) is a logical extension of the existing Network Access Machine. The implementation currently under way will accommodate field flagging (by either the user or the EAS if the information is contained in its database) and adaptive accommodation to unexpected system responses.

To accomplish these objectives, the EAS initially records all characters exchanged between a user and the system as demarcated by the user's overt initiation and termination of the recording session. A translator is then invoked to transform these character strings into proper NAM macros which may be called and expanded at any time. Variable fields are initially accommodated through flagging via a control character. Subsequently, as the NAM knowledge base is implemented, automatic recognition and flagging of variable fields (parameters) will be provided. As identified in Section 6.1, there are two approaches to handling unexpected responses. In view of the evident impossibility of foreseeing all possible system responses, the approach being adopted at NBS is the second (user intervention) coupled with preprogramming of some 'expected' unexpected responses. Thus, the EAS will have a minimal learning capability sufficient to permit incorporation of unexpected responses when they occur and, thereafter, to automatically handle such responses.

Usage of an expert assistance system by expert users constitutes an interesting, potential application. Users who are "expert" in the use of specific services on specific systems use the expert assistance system to generate the appropriate commands in the language of the access system to automate access to services for endusers. These "expert" users may be from a variety of areas—information retrieval, data base management, text processing—and are enabled by the assistance system to sit down and quickly create a library of commands to support predictable requirements of endusers. The "expert" users are shielded from learning the command language of the assistance system; they simply sit at terminals and enter the commands specific to their services. Further, the endusers are shielded from learning the command language of the services they wish to access and from learning the language of the assistance system.

A natural enhancement of an expert assistance system would be building access aids in the language of the access system interactively with the user. Thus, while the system is totally responsible for providing aids to support the inexperienced user, aids might be cooperatively generated while interacting with the more experienced user. Such a mechanism is discussed in the next section describing a highly interactive system in which the user is always being guided through the available alternatives.

DYNAMIC TUTORIAL ASSISTANCE

A dynamic tutorial system provides for user/service interactions controlled by the access system. The purpose of the tutorial system is to guide the user through all possible classes of activity offered by an access system. The tutorial system continually prompts the user and takes an appropriate course of action dependent upon the user's action. As an example of the use of the tutorial system consider the sample interaction below. (The italicized portions are generated by the user and the other portions are generated by the tutorial system.) This example terminates in the invocation of the required access system commands to login to the sending system, and transfer a file to the receiving system. The profile system contains the default values for the sending and receiving systems. If the user elects to use other than the default systems, the user may then have the profile updated with the new values at the end of the session.

File Manipulation? *yes*
Type of Activity? *transfer*
Sending Host is HOSTA? *yes*
Receiving Host is HOSTB? *yes*
Filename? *TESTER*

File transfer complete.

At this point the transfer is completed. This is of course the simplest sequence of events. Either the sending or receiving system could be other than the default ones; if so, the tu-

torial system may have to request user numbers and passwords for the new systems if they are not already known.

The essence of the dynamic tutorial assistance system is that the system is in command. It guides the user through all activities provided by an access system and executes the required commands to achieve the user goals. Additionally, the user could have the option to have the tutorial system catalog specific interactions so that the user may invoke them directly at a later time. Notice that this procedure implies a cooperative effort between a dynamic tutorial system and an expert assistance system. Using the above example, the user could invoke a "transfer" command which would result in the transfer of a file from one system to another.

SUMMARY

This paper has addressed issues relevant to implementing access support at the network level. An overview of the area of network access was presented, related research efforts were identified, factors which complicate network access support were identified, and the major components of network access support were structured. A specific implementation for one of these components, expert assistance, was described as it is being constructed at the National Bureau of Standards.

Our development of a structure for network access support began with the observation that user categories and access support difficulties preclude the existence of a single general support mechanism. Further, if the objective is provision of support via a network level group, only the acquisition, initialization and termination stages are reasonable candidates. This reflects the reality that service utilization requires intricate knowledge of the implementation of a specific service on a given system; it is unreasonable to expect one group (namely, the developers of a network wide access mechanism) to possess that level of knowledge for all systems on one network—much less across network boundaries.

Although support of service utilization cannot be provided at the network level, it is appropriate to provide general purpose support mechanisms. To accomplish this, an expert assistance system was introduced which can partially automate tailoring general purpose access mechanisms to meet individual objectives. Such a system shields users of access support systems from the command languages of those systems, while enabling them to construct procedures to access services on target systems.

ACKNOWLEDGMENT

The authors would like to express their appreciation to Rob Rosenthal and Sig Treu for their helpful comments and discussion of network access support.

REFERENCES

1. Pyke, Thomas N., Jr., "Some Technical Considerations for Improved Service to Computer Network Users," *Proceedings COMPCON 73,* February 1973, pp. 53-55.
2. Blanc, R. P., "Assisting Network Users with a Network Access Machine," *Proceedings ACM,* 1974.
3. Rosenthal, Robert, *Network Access Techniques—A Review, Proceedings 1976 National Computer Conference.* AFIPS Press, Montvale, New Jersey, 1976, pp. 495-500.
4. Rosenthal, Robert and Bruce D. Lucas, *The Design and Implementation of the National Bureau of Standards' Network Access Machine (NAM),* NBS Special Publication Series 500, in preparation.
5. Anderson, Robert H., "Advanced Intelligent Terminals As A User's Network Interface," *How to Make Computers Easier to Use* (COMPCON 75 Digest of Papers, Sept 9-11, 1975) IEEE N.Y., 1975, pp. 180-182, (IEEE Catalog No. 75ch0988-6c).
6. Fitzgerald, Mary Lynn, *Common Command Language for File Manipulation and Network Job Execution,* NBS Special Publication Series 500, in preparation.
7. Hendrix, Gary G., Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum, "Developing A Natural Language Interface To Complex Data," Artificial Intelligence Center Technical Note 152, August 1977, Stanford Research Institute, Menlo Park, California, 22p.
8. Hendrix, Gary G., "LIFER: A Natural Language Interface Facility", *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks,* May 25-27, 1977, pp. 196-201.
9. Marcus, Richard, "Network Access for the Information Retrieval Application," *Access to Computer Networks* (Session 25, IEEE Intercon 75, IEEE Intercon Conference Record) IEEE N.Y., April 1975, pp. 25/4: 1-7.
10. Treu, Siegfried, *A Testbed for Providing Uniformity to User-Computer Interaction Languages,* NBS Special Publication Series 500, in preparation.
11. Anderson Robert H. and James J. Gillogly, *RAND Intelligent Terminal Agent (RITA): Design Philosophy,* Rand Report R-1809-ARPA, February 1976.
12. Anderson, Robert H. and James J. Gillogly, "The RAND Intelligent Terminal Agent (RITA) as a Network Access Aid," *Proceedings of the National Computer Conference 1976,* 1976, pp. 501-509.
13. Blanc, Robert P., "Assisting Network Users with a Network Access Machine," *Proceedings ACM,* 1974, November 1974.
14. Rosenthal, Robert, and Shirley Ward Watkins, "Automated Access to Network Resources—A Network Access Machine," *Computer Networks: Trends and Applications,* IEEE Inc., New York, 1974, pp. 47-50. (Proceedings of the 1974 Symposium sponsored by the National Bureau of Standards, Gaithersburg, Maryland, and the IEEE Computer Society).
15. Marcus, Richard, *The Networking of Interactive Bibliographic Retrieval Systems,* Massachusetts Institute of Technology, Report ESL-R.-656, Cambridge, 164p.
16. Faneuf, Ross A., "The National Software Works: Operational Issues in a Distributed Processing System," *ACM 77 Proceedings of the Annual Conference,* Association for Computing Machinery, New York, 1977, pp. 39-43.
17. Geller, Dennis P., "The National Software Works—Access to Distributed Files and Tools," *ACM 77 Proceedings of the Annual Conference.* Association for Computing Machinery, New York, 1977, pp. 44-52.
18. Millstein, Robert E., "The National Software Works: A Distributed Processing System," *ACM 77 Proceedings of the Annual Conference,* Association for Computing Machinery, New York, 1977, pp. 53-58.
19. Heafner, John F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance,* Information Sciences Institute, University of Southern California, ISI/RR-74-21, ARPA Order No. 2223, September 1974, 53p.
20. Shortliffe, E. H., *Computer-Based Medical Consultations: MYCIN,* American Elsevier, New York, 1976.
21. Sacerdoti, Earl D., "Language Access to Distributed Data with Error Recovery," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* Cambridge, Massachusetts, August 1977, pp. 196-202.

22. Sagalowicz, Daniel, "IDA: An Intelligent Data Access Program," Artificial Intelligence Center Technical Note 145, June 1977, Stanford Research Institute, Menlo Park, California, 36p.
23. *ARPANET Resource Handbook,* the Network Information Center, NIC 39335 Augmentation Research Center, Stanford Research Institute, Menlo Park, California, December 1976.
24. Benoit, John W. and Erika Graf-Webster, "Evolution of Network User Services—The Network Resource Manager," *Computer Networks: Trends and Applications,* IEEE Inc., New York, 1974, pp. 21-24. (Proceedings of the 1974 Symposium sponsored by the National Bureau of Standards, Gaithersburg, Maryland, and the IEEE Computer Society).
25. Wyatt, Joe B. and Vincent I. Polley, "Network Interface Systems—An Evaluation by Simulation," *Proceedings of the National Computer Conference 1976,* 1976, pp. 511-521.

Adaptive random data generation for computer software testing

by STEPHEN F. LUNDSTROM
Burroughs Corporation
Carlsbad, California

INTRODUCTION

This paper discusses computer-assisted generation of input data for program testing. The test data generation system must produce input data that is per specification, unbiased by prior analysis of the program to be tested. The test data generators evaluated use random generation techniques to produce the test data. To reduce the number of sets of test data needed to test a program, summary information about the performance of previously generated sets of input data is used to modify the probability distributions upon which the next set of test data is based. Four test data generators were evaluated and used to generate test data to exercise five testcase programs of various complexity. Observations of the results of the actual evaluation runs and of the types of structures involved led to the establishment of some guidelines for future testing operations. Program verification of any sort was not attempted. Manual checks of the normal program outputs did detect a number of software errors. No attempt was made to automatically isolate or even detect software faults. Nor was any comparison with other methods, manual or automatic, made.

The Software Testing System developed to use as a tool to evaluate the test data generators is similar to others previously reported.[1-7] These systems require repetition of steps to guide and assist test case selection, to execute the instrumented program and to analyze testing coverage until the test goals have been achieved. The system used here to evaluate the test data generators not only provided the summary reports of testing progress similar to the above systems, but also provided progress summary information to the test data generators which was then used to modify the probability distributions controlling the generation of the test data.

Most previous work relating to the generation of test data for program testing in some way base the test data on analysis of the program to be tested. R. L. Sauder[8] determined the format and description of the test data by analysis. Other systems[9-12] use analysis of the program to guide choice of test data when the main goal of testing was usually to execute all paths through a program.

Program analysis is subject to any related programming errors in the program to be tested which may, in turn, cause the generation of test data which is inconsistent with the original specification. If any paths are missing in a program to be tested, they would not normally be detected through use of test data generators which are based on program analysis.

Program test data should be based, as much as possible, on the program specifications rather than on the program implementation. Sauder's work was unique in that it required the user to specify the data relationships thus allowing the "semantics" of the test data to be derived from the specification and whatever test plan might exist. The systems reported by W. H. Burkhart,[13] and K. V. Hanford[14] are specifically concerned with testing compilers and generate test data from the formal syntax of the language to be processed. These systems are unable to take any semantic information, such as relationships or limitations between syntactic variables, into account in the generation of test data. A new program product recently introduced by Burroughs Corporation[15] does allow specification of both the desired syntax and semantics of the desired test data. Each of these three systems generate data based on a uniform random distribution generator without any concern as to the efficiency of such an approach.

The only reported work involving random test data generation where techniques were evolved to improve the efficiency of the generators was in the area of hardware test data generators used to test digital logic.[16-17] The original concepts have been further extended by K. P. Parker.[18-22] Parker's system allows the probability of a particular value of any test input to be based on the average or most probable value of previously successful sets of test data.

Although the hardware techniques were developed without regard to software problems, there are many areas in common and some of the hardware experience can be applicable.[23] Although the concept appears valid, hardware testing is the process of searching for equipment malfunction (assuming a correct design), but software testing is the process of somehow verifying the design and implementation of the software itself.

SOFTWARE TESTING SYSTEM

The software testing system used for evaluation of the test data generators is organized into two phases as shown in Figure 1. The first phase, program analysis, inserts monitors in the program to be tested. The monitors allow observation of flow through the control structure of the program, and also observation of changes in the values of variables during execution. The second phase, program test and evaluation, is the phase which generates test data, runs the instrumented program and evaluates the results of the tests. In each of the repeated cycles, a new set of input data is produced, the program (with monitors) is executed, and the results are evaluated. Three of the four test data generators evaluated were adaptive and utilized monitor-generated summary information relating to the success of previous tests in their heuristic algorithms.

The actual format and values of the input data must be as specified for the particular program under test. Special programs supported this function for each of the five test programs used. In a production environment, a special input data specification language would be developed together with a processor for that language. Even though each program to be tested has special requirements for the form and range of its input data, the procedures used for selection of the data are generalized and apply regardless of the program to be tested.

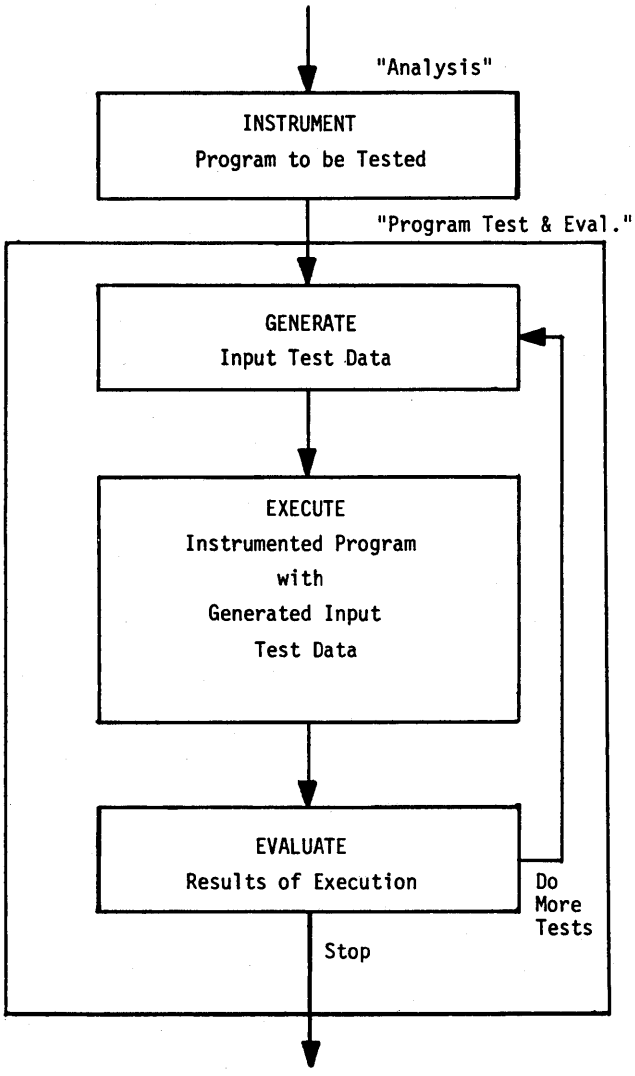


Figure 1—Software testing system

Monitoring requirements

When developing a testing system of any sort, it is important to know what the test procedure has tested and to what extent. This information is important both in determining what is faulty if a problem is found and in determining when sufficient testing has been done. The system developed monitors both the control structure and range of variables as a means of measuring the ability of the generators to "thoroughly" test the software.

Branch monitoring

All possible branches within the program graph are monitored. The obvious goal is to attempt to exercise all possible branches. Even if all possible branches are exercised, all possible paths in the program being tested may not have been exercised. In addition, there may be paths which have not yet been implemented and which cannot be monitored. The simpler measure of branch executions was chosen as one measure useful in demonstrating the viability of the proposed adaptive random data generators.

Range of value monitoring

The user may define a desired range of values to be monitored for each variable of interest. Each range is divided into "N" equal size segments (another simple measure chosen as sufficient to demonstrate the viability of the generators, in this case $N=10$). Assignments to these variables are monitored with the goal of observing data values in each of the segments of the range for each variable. The specified range of interest is not required to be the same as the observed range of values for any variable.

Termination conditions

The need for well-defined termination conditions are particularly important in a testing system where there may be no guarantee that all testing requirements can ever be met. The following conditions were adopted, any of which can force immediate termination of any testing in progress:

1. "M" test data sets have been applied to the program being tested. (M=50 was used.)
2. All possible branches have been observed to have ex-

ecuted and values have been observed in each segment of the range of values for each monitored variable.

3. The last "P" test data sets were inefficient. In this case, if P=10 successive test data sets were not productive, termination would occur. A test data set was considered not productive if the sum of "new" branch executions observed and "new" range of value segments observed is "R" or less (R=2 during testing). "New" branches are those observed which had not been executed previously; "new" segments are those in which values were observed which previously had not been.

Software test sequence

The actual procedure of software testing can start after the program analysis phase has generated the instrumented source program. The system developed (see Figure 2) consists of a central control which, after any necessary initialization, sequences through automatic test data generation, execution of the instrumented source program and test evaluation. The conditions which cause termination were defined previously. These conditions are checked during the test evaluation phase. If termination is not yet required, the central control restarts the test generate, execute and evaluate sequence. The automatic test data generators are discussed in detail below.

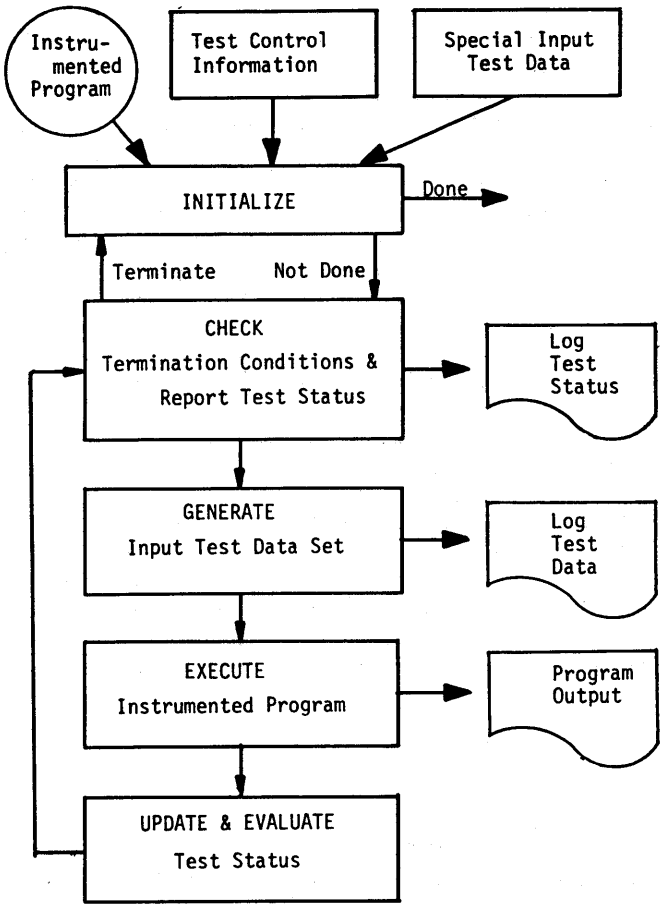


Figure 2—Software testing system—Program test and evaluation phase

Manual verification

The software testing system developed is not capable of automatic verification of the validity of the outputs of a program given the automatically generated inputs. The system produces summaries of the results of all testing (including which branches have not been executed and what range of value segments have been observed). This information together with the normal program outputs must be manually inspected as the final step of program testing.

AUTOMATIC TEST DATA GENERATION

The major problem faced in automatic test data generation is how to devise adequate tests. Program development proceeds through a sequence of four steps; specification, design, implementation, and verification. If the specification is assumed correct, then the design and implementation should be checked. The test data generation system described below generates input data based on the specification. It also allows use of manually generated or previously saved input data. By monitoring the execution of the program, the testing process covers both the design and implementation. Other systems, which generate test data based on analysis of the program being tested only have the capability of exercising the implementation. Analysis-based test data generators produce test data with implementation errors reflected in the test data. Therefore, many kinds of implementation errors are masked. Specification-based test data generation does not have this problem. Since input data is based directly on the specification, any errors due to the design or implementation procedure are vulnerable to detection. Errors such as missing control paths or incorrect handling of input data are likely to be observed using specification-based test data.

In a production-oriented software testing system, an input data definition language should be available to allow the representation of the portion of the program specification which defines expected input data in an unambiguous, machine-recognizable form. In the system implemented, that portion of each generator which would normally interpret an input data description was specialized for each of the programs tested. The specialization was based on the specification of the input data to be produced and not on analysis of the programs tested.

Recall that the software testing system monitors both the sequence of execution and range of value segments. The sum of the number of "new" branches and "new" range of value segments (where "new" indicates previously unobserved) is used as a means of comparison of the productivity of two test data sets and is used during evaluation of the termination condition.

The judicious selection of a small number of input test

data sets from an extremely large domain is another major problem when developing an automatic test data generation system. The capabilities of four input test data generators were evaluated. The four generators are called UNIFORM, GAUSS, INVERSE, and NEXT-BEST. All of these generators, except UNIFORM, are designed to adapt to the most effective areas of the input data space.

UNIFORM test data generator

The UNIFORM test data generator is the only generator evaluated which is not adaptive. Values are chosen randomly over the given legal range of the particular test data element being chosen. A uniform probability distribution guides the choice so that each value in the range is equally likely to be chosen as any other value in the range. In addition to allowing direct use of the UNIFORM generator, each of the other generators use the UNIFORM generator under circumstances explained below.

GAUSS test data generator

The development of the GAUSS test data generator was motivated by the observation that many program organizations are not symmetric. The simple example below illustrates the asymmetry. The example procedure (Figure 3) receives an input parameter X which is an integer in the range of 0 through 100. If X is less than 50, a parameter Y is incremented. If X is less than 50 and is even, then a parameter Z is multiplied by X. If input data to test the example procedure is chosen with a uniform distribution, one half of the cases, on the average, will test only the first conditional. Only one quarter of the cases, on the average,

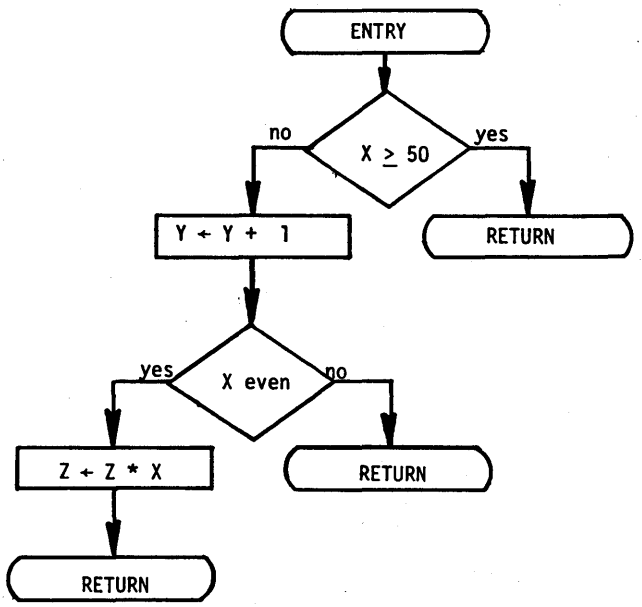


Figure 3—Flowchart of example showing control asymmetry

test both the code which increments Y and which multiplies Z by X. If those were major program blocks, it is clear that much of the work of a UNIFORM test data generator would be ineffective in testing the example program. The asymmetry also exists in computation of variable values. The GAUSS test data generator is designed to bias choice of test data by consideration of previously useful test data sets.

Test data generators are called productive if the data produced exercise portions of the program not previously tested. In terms of the productivity requirement, the program organization (or at least that part of interest) is dynamic. After each application of a set of inputs, that portion of the program which actually was exercised "disappears" and is no longer used in evaluation of the performance of the generator. The goal of the generators is to produce test data which exercises the "remaining (untested) portion" of the code.

Once the generator produces inputs which force execution of a branch of the program not previously executed, the generator becomes productive. The productivity might be maintained by choosing inputs to again enter the same branch, but perhaps test a different subbranch. One can usually assume that entry to a particular branch is controlled by some subset of the input variables. Thus, if the proper input variables are kept "the same" and others are modified, it is likely that various subbranches will be exercised. However, no analysis is performed which would determine the subset of inputs which control entry of some branch. If input values are chosen randomly under control of a Gaussian distribution, and if the standard deviation is small, and if the mean represents values needed to enter the branch, the generation of values the same as before becomes likely. The variation in values should be limited enough to continue generation of tests of a productive branch but, at the same time, be sufficient to exercise other subbranches. Eventually, as all the subbranches have been exercised, the productivity of test inputs based on that distribution will decrease. As this occurs, other successful test sets will be used to compute the distribution, and inputs will be produced to test other portions of the program.

Initial GAUSS distribution

Unless the user provides guidance, the GAUSS test data generator has no means of knowing where to set the mean and standard deviation initially. In order to provide some initial history, the GAUSS generator produces the first "C" cases with the UNIFORM generator. For purposes of evaluation, "C" was arbitrarily chosen to be four. The UNIFORM cases are not generated if any manual or previously generated cases are input initially.

GAUSS generation technique

The GAUSS generator bases the test data produced on a Gaussian probability distribution. Each variable generated with GAUSS has a corresponding mean and standard deviation which, together with the defined range of the variable,

control the values produced. The mean and standard deviation for each variable are not fixed. These parameters are adaptive. Each time the GAUSS generator is activated, the mean and standard deviation of each variable is recomputed based on the latest "successful" tests.

It remains to be determined which tests are successful. First, a test set is assumed to have certain productivity, defined for purposes of evaluation to be the sum of the number of "new" branches executed and the number of "new" variable value ranges observed. Recall that "new" simply means "not previously observed." The test data chosen to compute the means and standard deviations are those whose productivity exceeds the average productivity of the "M" most recent sets of test data.

The size of the collection of test data sets considered, "M", also is adaptive. If a number of successive data sets are not productive, the range of "M" is doubled up to an arbitrary limit. On the other hand, if a number of successive data sets are productive, the range of "M" is halved, but not less than the original value (in this case, 5). The size of the collection of data sets used to define the variable value distributions varies because of many of the arguments summarized earlier. A period of high productivity can often be maintained by basing distributions on a small number of recently successful tests. A period of low productivity is often characterized by a search for another productive area. A search of this sort is often successful when not biased by only a few sets of previously successful data, but by a broad representation of all previously successful tests.

The standard deviations of the distributions used to produce the test values vary widely. When a particular value V_0 of a variable V is useful to force control into a particularly productive set of tests, the standard deviation of V may become very small as more and more tests with values of V near V_0 are produced. If the value of a particular data element is not important in that way, the standard deviation is likely to be very large, often large enough that the Gaussian distribution over the legal range of values is essentially uniform.

INVERSE test data generator

The development of the INVERSE test data generator was motivated by impatience with the dynamics of the GAUSS generator. In particular, because of the procedures used to compute the distributions of the various input data values, a number of test sets would be produced before the deviation was large enough to allow an effective search for new productive areas. The problem appeared to be that data generated based on current distributions was often non-productive and no reliable information was available as to which part of the remaining portion of the input space would be productive.

INVERSE and GAUSS commonality

The resulting INVERSE data generator operates the same as the GAUSS generator during computation of variable value distributions, establishment of an initial history with cases produced by the UNIFORM generator, adjustment of the size of the collection of test cases used for distribution computations, and random generation of data based on a Gaussian distribution. Only after the normal GAUSS procedures become non-productive is the INVERSE generation technique activated.

INVERSE generation technique

The only difference between the GAUSS and INVERSE generators is how data values are produced after non-productive test data sets. If the previous test data set is not productive, the INVERSE search procedure is activated to attempt to find a new productive region in the input space. Successive activations of the search procedure alternate between standard generation of input data with the GAUSS generator and generation of input data with the INVERSE procedure. A true inverse distribution is not really appropriate. The inverse GAUSSIAN distribution would tend to generate probable values at the boundaries of the input data space. Rather, it is desirable to produce values not likely to be produced with the GAUSS generator, but still provide an unbiased search of the rest of the input space. The UNIFORM generator can accomplish this easily with the addition of a simple test of each of the variables generated. If a UNIFORM generated value is in the "GAUSSIAN-likely" range (MEAN−STANDARD DEVIATION through MEAN+STANDARD DEVIATION) where MEAN and STANDARD DEVIATION are as normally computed for the GAUSS generator, then another replacement value is produced (through use of the UNIFORM generator once again). When none of the generated values lie within the "GAUSSIAN-likely" range, the data values are output and applied to the program being tested. Analysis of the productivity and recomputation of the variable distributions then proceed as previously described with GAUSS. In many respects, the INVERSE test data generator can be thought of as a random data generator controlled by a UNIFORM distribution which has a hole in it, centered at the variable MEAN and with radius of the STANDARD DEVIATION. It allows a uniformly distributed random search of the area not recently productive.

NEXT-BEST test data generator

The NEXT-BEST test data generator was motivated by the observation that once a particularly productive input data set is chosen, other productive data sets can often be produced solely by changing only one of the variables in the input data set. Calculation of most variable values can be considered to be some function of the input variables. Variations of only one input variable will usually affect the computed values of some of the program values. Similarly the expressions which control branches between basic blocks in the program are also functions of the input variables. The appropriate modification of only one input variable

can cause the execution of different basic blocks. The effect of modifying only one input variable is essentially an orthogonal search of the input data space for other productive cases given a known productive case. Any arguments in favor of this technique for execution of basic blocks is equally valid for testing ranges of variables.

Initial NEXT-BEST distribution

The NEXT-BEST test data generator must have at least one set of input data values before modification of individual, randomly-chosen input values can be done. If no manual or previously specified data sets are input initially, one input data set is produced using the UNIFORM generator. If more than one manual or previously specified data set is input, the most productive will be used during the next phase of the NEXT-BEST procedure, input variable distribution initialization. The case chosen will be called the base case.

Input variable distribution initialization

The input variables which are randomly selected will not be equally productive during the use of the NEXT-BEST generator. In an attempt to further improve efficiency, the most productive variables are given a somewhat greater probability of being used to generate new data sets under the implementation of the NEXT-BEST generator. Before normal processing begins, each of the input variables is used to generate at least one input data set which is a modification of the base case chosen after the Initial Distribution phase. If the data set generated by modification of a particular input variable is productive, then the same variable will be used again. If non-productive, the next variable is used. The values generated for each of the chosen variables are produced through use of the UNIFORM generator. A record of the total productivity of each variable is kept both during this and the final phase. The productivity information is utilized during input variable selection.

NEXT-BEST generation technique

After Initial Distribution and Input Variable Distribution initializations, the major test data set generation phase begins. This phase consists of a number of subsequences of test data set generation steps. Each subsequence begins with the choice of a base case. The new base case chosen is that test data set previously generated which has the largest productivity of all those not yet used as a base case. Upon completion of a subsequence this base case is marked as used.

After a base case is chosen to start a subsequence, a new input variable is chosen. The choice is made randomly but with a preference toward previously productive variables. To make the choice, an assignment of each variable to a portion of the range of real numbers from 0 through 2 is made. Let S_i be the size of the portion of the range assigned to the ith variable. Assume there are n variables and that P_i is that fraction of the total productivity to date contributed by the ith variable. Then:

$$S_i = \frac{1}{n} + P_i .$$

A random number between 0 and 2 is then chosen under control of a uniform distribution. That number will be within the portion of the range from 0 through 2 assigned to one of the variables and will be used to select that variable. The order of the variables across the range does not vary and is the same as the order of selection during variable distribution initialization phase.

Once a variable is chosen, a new value is selected for it to replace its original value in the current base case. The selection is under control of the UNIFORM generator. If the resulting test data set is productive, the same variable is used again for production of the next data set. When a data set is not productive, the next variable in the same order as chosen during the variable distribution initialization phase is selected. By sequencing to other variables in this way, variables with poor past productivity can develop a more complete history of productivity. The subsequence ends when the end of the sequence of variables is reached. At that point, the base case is marked as used and a new subsequence is initiated.

EVALUATION AND RESULTS

The software testing system together with the automatic test data generators were evaluated through use of some programs chosen as testcases. Limitations of manpower and computer time reduced the number of programs chosen to five. They were chosen to represent a wide variety of program application and complexity. Three of the programs are student submissions from an introductory programming course and represent table lookup, search techniques, interpolation, numerical approximation, divided difference, boundary value and game playing applications. In these three cases, the general problem was chosen as representative and one of the programs submitted was arbitrarily chosen for each. The other two programs chosen are production programs written (and selected) by a staff analyst other than the author. These two programs are numerical analysis programs concerned with the analysis of stresses in turbine blades. The five programs were called:

Temperature Distribution in a Conducting Solid
The Series Magnetic Circuit with an Air Gap
Tictactoe (a tournament)
Aerospace Analysis
Turbine Blade Analysis

(See Lundstrom[24] for a more complete description of each of the five testcase programs.

Test procedures

Each of the five testcase programs was "tested" by each of the four test data generators. Four of the testcases were also "tested" with at least one prespecified set of test data values. In order to gather reasonable statistics, each of the procedures above was performed ten times.

Measurement and observation of results

Control flow and range of value monitors were chosen since they reflect, at least to some degree, the thoroughness of a collection of test data sets. In the future, other measures may prove more useful. The monitors are observed and recorded after each test data set is applied to a testcase program. This information, when averaged over the ten times each generator was used to test each testcase program, gives a good feeling for the expected performance of each generator.

Summary of results

The software test system proved to be useful in detecting errors in the programs tested. More importantly, even though the input test data generators used cannot be guaranteed to find all errors, the generators produce unbiased test data which is more likely to thoroughly exercise a program, with respect to its specification, than its creator would.

The NEXT-BEST generator, as implemented, never demonstrated productivity better than the others. The results might have been different if more initial cases were produced, if more initial variable productivity history development was performed, and if variables known to be nonproductive were used less during the main body of the procedure.

The GAUSS generator proved to be most successful in situations where a very non-uniform mapping from input data space to the program occurs such that only one productive region exists. In similar cases with more than one productive region, the INVERSE generator was most productive, since distribution repositioning can be speeded through the UNIFORM "search."

The UNIFORM generator represents the type of test data generator most used in the past and is most productive when the mapping described is uniform. If the mapping is not known, then the UNIFORM generator may be slightly more productive than the INVERSE generator, most likely due to an insufficient "UNIFORM" search for productive areas before the main INVERSE procedure begins. Although the INVERSE generator is never quite as productive as the UNIFORM generator when the mapping is uniform, it is sufficiently better when a non-uniform mapping exists (especially if a good manual case is input) that the INVERSE generator can be recommended as the generator to use in the many situations when the mapping is not known.

Recommendations

An improved generator, called COMPOSITE, has been considered, but not evaluated. It should offer better early productivity and less sensitivity to the mapping. COMPOSITE would make use of both the UNIFORM and INVERSE generators described. However, COMPOSITE would allow selection of the generator to be based on past performance. The mapping changes for each variable as more and more of the program is exercised. By allowing the data generator to reflect the current mapping, an increase in productivity should be observed. Additional suggested features of COMPOSITE are described in Lundstrom.[24]

The values of many variables cannot be monitored realistically with the fixed-size, fixed-number range of variable value divisions utilized in the system implemented. It should be possible to specify the number of range segments together with the size of each segment (without the constraint that each segment be the same size). It is more important that the possible values be distributed uniformly across the segments of the range of values of a variable than to have equal-sized segments.

CONCLUSION

The goals of the work described here were to study random test data generation techniques as applied to software testing, to develop and evaluate random data generation algorithms, and to develop a complete software testing system to support the study. The software testing system was successfully implemented. The test data generators were evaluated by using them to test each of five programs chosen as testcases; programs which represent a wide variety of applications.

The four test data generators developed and evaluated were called UNIFORM, GAUSS, INVERSE and NEXT-BEST. While each of these was observed to be productive in some situations, only INVERSE was consistently productive. It was determined that knowledge of the type of mapping from the input space to the program could allow choice of the appropriate generator for most efficient operation. Since understanding of the mapping is often difficult or time-consuming, and since the system developed did not provide tools to support identification of the mapping, the INVERSE generator was determined to be the most broadly useful over the cases studied. An improved generator, called COMPOSITE, was proposed, but not evaluated.

Although program verification was not attempted, enough program faults were manually observed in the test output to demonstrate that input test data generation can be successfully based on the specification rather than on analysis of the program.

ACKNOWLEDGMENTS

The work described in this paper contributed to the fulfillment of the requirements of the Doctor of Philosophy degree while the author was enrolled at Texas A&M University. Dr. Dick B. Simmons provided motivation to investigate the importance of range of variable value monitoring as well as much needed encouragement throughout. Dr. Richard E. Fairley was the catalyst leading to the understanding of how the relationship between the input test data set domain and the program affects the choice of the test data generator. Computer facilities were provided by Texas A&M University and Burroughs Corporation.

BIBLIOGRAPHY

1. General Research Corporation, *RXVP FORTRAN Automated Verification System, Level 1 Reference Manual,* May 1975.
2. McDonnell Douglas Automation Company, *Program Evaluator and Tester.*
3. Stucki, L. G., *A Prototype Automatic Program Testing Tool,* MDAC Paper WD1985, McDonnell Douglas Astronautics Company—West, August 1972.
4. Stucki, L. G., "Automatic Generation of Self-Metric Software," *Record of 1973 IEEE Symposium on Computer Software Reliability,* 1973, pp. 94-100.
5. Ramamoorthy, C. V., R. E. Meeker, Jr. and J. Turner, "Design and Construction of an Automated Software Evaluation System," *Record of 1973 IEEE Symposium on Computer Software Reliability,* 1973, pp. 28-37.
6. Russell, E. C. and G. Estrin, "Measurement Based Analysis of FORTRAN Programs," *1969 Spring Joint Comput. Conf., AFIPS Conf. Proc.,* Vol. 39. Montvale, N.J., AFIPS Press, 1969, pp. 723-732.
7. Fairley, R. E., "An Experimental Program Testing Facility," *IEEE Transactions on Software Engineering,* Vol. SE-1, pp. 350-357.
8. Sauder, R. L., "A General Test Data Generator for COBOL," *1962 Spring Joint Comput. Conf., AFIPS Conf. Proc.,* Vol. 21, Montvale, N.J., AFIPS Press, 1962, pp. 317-323.
9. Krause, K. W., R. W. Smith and M. A. Goodwin, "Optimal Software Test Planning Through Automated Network Analysis," *Record of 1973 IEEE Symposium on Computer Software Reliability,* 1973, pp. 18-22.
10. Huang, J. C., "An Approach to Program Testing," *ACM Computing Surveys,* Vol. 7, Sept. 1975, pp. 113-128.
11. Howden, W. E., "Methodology for the Generation of Program Test Data," *IEEE Transactions on Computers,* Vol. C-24, May 1975, pp. 554-559.
12. Miller, E. F. Jr., and R. A. Melton, "Automated Generation of Testcase Datasets," *Proceedings 1975 International Conference on Reliable Software,* 1975, pp. 51-58.
13. Burkhardt, W. H., "Generating Test Programs from Syntax," *COMPUTING,* Vol. 2, New York, Spring-Verlag, 1967, pp. 53-73.
14. Hanford, K. V., "Automatic Generation of Test Cases," *IBM Systems Journal,* Vol. 9, 1970, pp. 242-257.
15. Burroughs Corporation, *Test Data Generator, Program Product Development Aid Specifications,* April 1977.
16. Chang, H. Y., E. G. Manning, and G. Metze, *Fault Diagnosis of Digital Systems,* New York, Wiley-Interscience, 1970.
17. Breuer, M. A., "Generation of Fault Detection Tests for Sequential Circuits," *Digest of 1971 International Symposium on Fault-Tolerant Computing,* IEEE Computer Society Publ., 1971, pp. 18-21.
18. Parker, K. P., *Monte Carlo,* Illiac IV Offline Diagnostics Group, Memo DN-236, November 30, 1971.
19. Parker, K. P., *Offline Diagnostics for Illiac IV, A User's Guide,* Moffett Field, CA, IAC, Ames Research Center, 1972.
20. Parker, K. P., *Adaptive Random Test Generation,* Technical Note 73, Stanford, CA, Digital Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, October 1975.
21. Parker, K. P. and E. J. McCluskey, "Probabilistic Treatment of General Combinatorial Networks," *IEEE Transactions on Computers,* Vol. C-24, June 1975, pp. 668-670.
22. Parker, K. P. and E. J. McCluskey, *Sequential Circuit Output Probabilities from Regular Expressions,* Technical Report 93, Stanford, CA, Digital Systems Laboratory, Stanford Electronics Laboratory, Stanford University, June 1975.
23. Hecht, H., "Can Software Benefit from Hardware Reliability Experience?" *IEEE 1975 Annual Reliability and Maintainability Symposium, Washington, D.C.,* Jan. 1975, pp. 28-30.
24. Lundstrom, S. F., *Adaptive Random Data Generation for Computer Software Testing,* a Dissertation, College Station, TX, Texas A&M University, May 1977.

Area Director:
Richard L. Nolan
D. P. Management Corporation
Lexington, Massachusetts

DP management and administration

OVERVIEW

The area of data processing management and administration covers topics related to the effective utilization of data processing personnel and technology to help achieve the objectives of the organization. Personnel costs continue to grow faster than hardware costs, and technology improvements continue to encourage more widespread uses of data processing. How our methods of data processing management and administration can help us cope with changes such as these is theme of the sessions in this area for NCC 78. The sessions can be grouped around two general issues. The first issue is the increasing influence of people management on the effectiveness of data processing. The sessions related to this theme include: time management, motivations, and why is D. P. management so difficult? The second issue is the impact of new techniques and technology on DP management and administration. Related to this general issue are sessions on: information technology and organizational response, security assessment techniques, contribution of planning to information systems productivity, DP auditing, and organizational considerations in the allocation of computer resources.

Time management for the data processing professional

by J. F. TOWSEN
Harrisburg, Pennsylvania

No longer can the Technical Professional ignore a day-to-day interface with peer groups from within the Organization.

The advent of newer—smaller and faster—operating systems have made the user a new partner in the development of the Corporate Data Processing Strategies.

But with this new role of partner came a new and different type problem for our technical professional. A new problem called TIME MANAGEMENT.

When they started to become more involved in the BIG PICTURE, they started just like their peer groups to complain about:

Meetings	Telephone
Interruptions	Postponed Decisions
Lack of Planning	Crisis Management
Visitors	No Priorities
Snap Decisions	Peer Demands on Time

The story is always the same, someone or something is WASTING my TIME. But have you noticed how most people are reluctant to admit their own faults or shortcomings as the real TIME WASTERS.

Telephones and meetings are very important tools for the D. P. Professional. Ideas need to be exchanged, questions asked or answered.

And let's not overlook the team members of this new partnership who can't COMMUNICATE. Behind most every user-D. P. problem, is a breakdown in some form of communications. When will they ever realize how much time is lost as a result of poor communications, and TIME IS MONEY.

Up to this point I have dwelt on the Data Processor. But we must understand that they are no different than other managers, most of whom don't know how to MANAGE their OWN TIME.

How many times have you seen major Data Processing projects go down to the wire or even come in late? Why does this happen? Why is all of this time and money wasted because of unnecessary reruns?

There are many reasons, but I do suggest that part of the problem comes from the D. P. Manager's inability to manage the many demands on their time. The normal management functions such as planning, delegation, follow-up and communicating get lost in the fast pace of doing business.

And we can't forget the cluttered desk and cluttered office. I believe in the saying "CLUTTERED DESK EQUALS CLUTTERED MIND." I have seen MANY D. P. centers and for the most part they are cluttered.

Along with many other Managers, the D. P. Professional must develop some of the basic techniques of good TIME MANAGEMENT.

This seminar will help the manager to develop some of these techniques. We will consider some of the following:

1. Spend more time on FIRE PREVENTION and less on FIRE FIGHTING.
2. Spend 10 or 15 minutes reflecting on the day's activity and planning for tomorrow.
3. Prepare a THINGS TO DO list detailing the major items to be accomplished tomorrow.
4. Don't become involved in areas of responsibility directly assigned to others.
5. Consider a management by exception program. Only the important exceptions reach your desk and require your time, energy and ability.
6. LEARN TO SAY NO.
7. If your job entails dealing with outsiders, learn to control these interruptions. Always be in control of the situation.
8. Don't spend all of your time "getting organized" only to find at the end of the day that nothing was accomplished.

And as part of the same seminar, we will look at how to get more out of our meetings, and how to "get an extra hour out of every day."

Organizational response and information technology*

by RICHARD L. NOLAN
Richard L. Nolan, David P. Norton, and Company, Inc.
Lexington, Massachusetts

INTRODUCTION

In one instance after another, organizations have remolded themselves, not in direct response to great ideas, but in response to the development of intervening technology that stimulates implementation of those ideas.[1] The remolding, however, does not immediately correspond with the advent of the new technology. An organizational response process is initiated with the advent of the new technology, and then evolves over time. The purpose of this paper is to render theoretical frameworks that are useful for understanding the organizational response process and then to apply the frameworks to add perspective to contemporary organizational developments in respect to information technology.

RELEVANT THEORY

There are two theoretical frameworks that are particularly relevant to the impact of information technology on organization. The first is an organizational change framework developed by Harold J. Leavitt. The second is an organizational learning framework developed by Richard L. Nolan.

Organizational change framework

Leavitt's organizational change framework views the organization as a complex system consisting of four main variables: task variables, structural variables, technological variables, and human variables. As shown in Figure 1, the variables are interrelated. A change in one results in compensatory change in the others. Usually, efforts concentrate on structure, people, or technology variables in order to effect an improvement in the task variable, but other variables also react to the change.

For example, a structural change such as decentralization may have been made with the primary intention of affecting the way in which production tasks are carried out. However, the change will most likely have an impact on the technology, such as a shift toward distributive data processing from centralized data processing. The impact on the technology variable could be consciously intended, or could be unforeseen and often a costly outcome of an intention to change only one variable.

Recognizing that a change in one variable affects the others, the remainder of this paper will concentrate on the effects of developments in information technology and the resultant impact on structure variables or, more specifically, organizational response. Information technology is defined as any technology that affects the way in which people in organizations carry out decision-making and administrative activities. The essential concept is that new information technology enables implementation of ideas about more efficient organizational structures. This flow from new information technology can be better understood by applying a second framework.

Organizational learning about information technology framework

Nolan's organizational learning framework views the assimilation of information technology as an organizational learning process involving four growth process variables: application portfolio variables, DP (data processing) organization variables, DP planning and management control system variables, and user awareness variables. The applications portfolio consists of all the automated systems of the company. It changes over time in a fairly predictable pattern. The other three growth process variables are acted on in order to develop an effective applications portfolio.

In Nolan's original work he identified four stages of maturity in the evolution of assimilating computer technology. The four stages are initiation, contagion, control, and maturity. The assimilation of data base technology requires four similar evolutionary stages. Typically, the data base initiation stage has coincided with the control stage of computer technology assimilation. The overlapping of the two four-stage evolutionary patterns produces the six stage evolution of the data resource functions illustrated in Figure 2.

* The author acknowledges helpful reviews of the ideas and concepts in this paper by William E. Bowen, David F. Dantzig, Benjamin S. Porter, Larry G. Robbins, and Gonzalo Verdugo, all of Richard L. Nolan, David P. Norton, and Company, Inc., Lexington, Massachusetts.

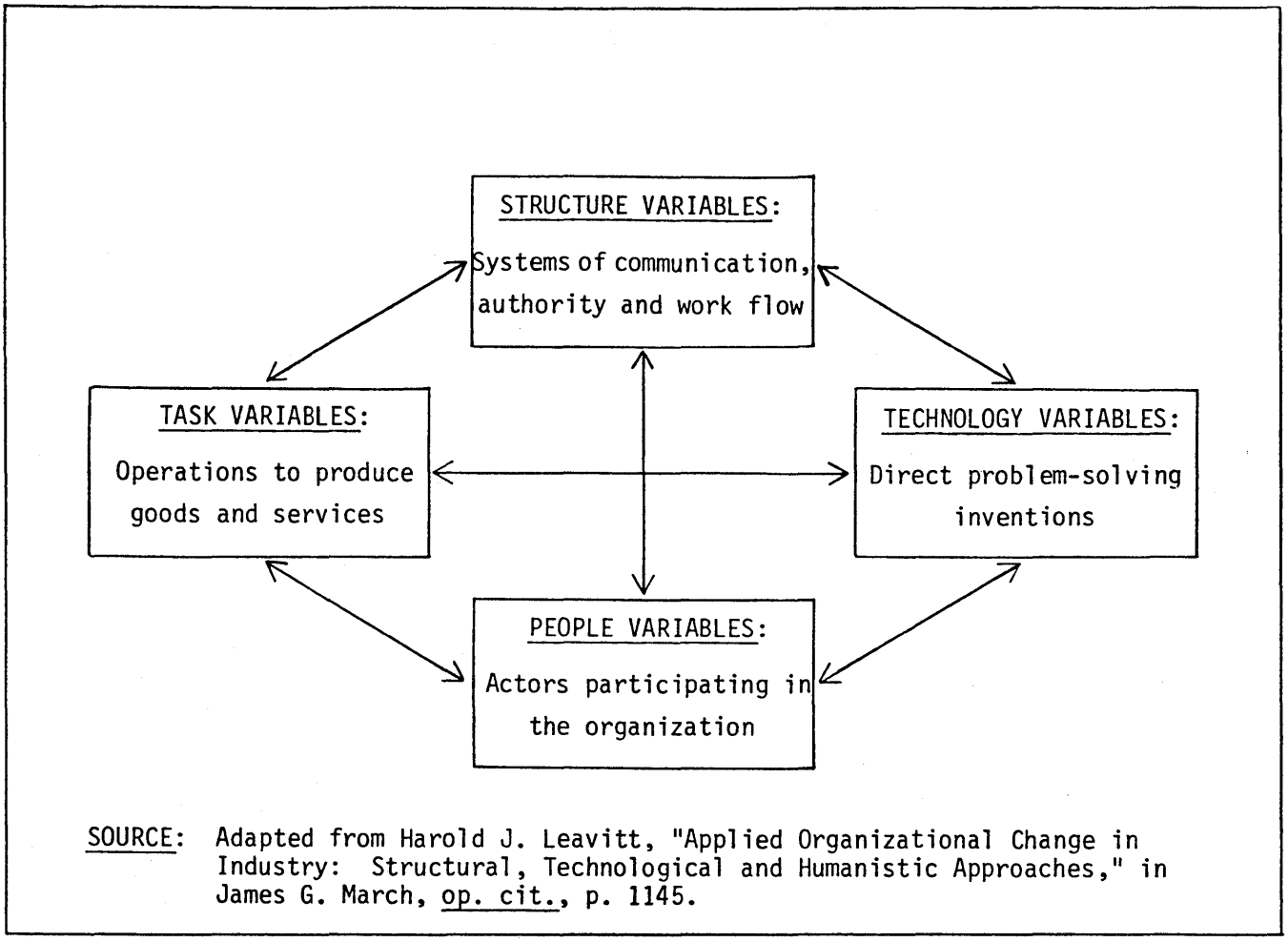


Figure 1—Leavitt organizational change framework

It should be noted that while data base technology is typically introduced during Stage III of computer technology assimilation, the degree of computer technology assimilation is only one of several factors affecting the timing of its introduction. Other factors affecting the timing are: (1) availability and maturity of the technology itself, (2) external knowledge of the technology, (3) business needs of the organization, (4) economic climate of the organization, and (5) competitive pressures for data-oriented customer services. Companies that are currently in Stage I or early Stage II of computer technology assimilation may well initiate data base technology in an earlier stage of maturity than has been typically observed in the past.

Nevertheless, the pattern of the data processing budget curve is a manifestation of the organizational learning. Stages I and II are predominately influenced by the assimilation of computer technology; Stages V and VI are predominately influenced by the assimilation of data base and data communication technologies. Stages III and IV are influenced by both types of technologies.

There are two key concepts that affect the rate and the way a company progresses through the six stages: (1) status of internal and external bodies of knowledge, and (2) balance between control and slack.

1. *Status of Internal and External Bodies of Knowledge*—At any point in time, there exists an external or professional body of knowledge and an internal, or company, body of knowledge on how to effectively manage information technology. The external body of knowledge is largely codified knowledge. It exists in books, seminars and courses. The internal body of knowledge is within the company and exists in documents and policies, as well as in the minds and behavior of the company's management. The internal body of knowledge is largely knowledge that is obtained through experience with information technology. It is also obtained through books, seminars and courses. But the critical point is that experiential knowledge within the company is the major factor in progression through the stages. Figure 3 illustrates the bodies of knowledge concept. The impact of external knowledge on the internal body of knowledge helps explain why companies that automated their first systems in 1970 move through the stages differently than companies

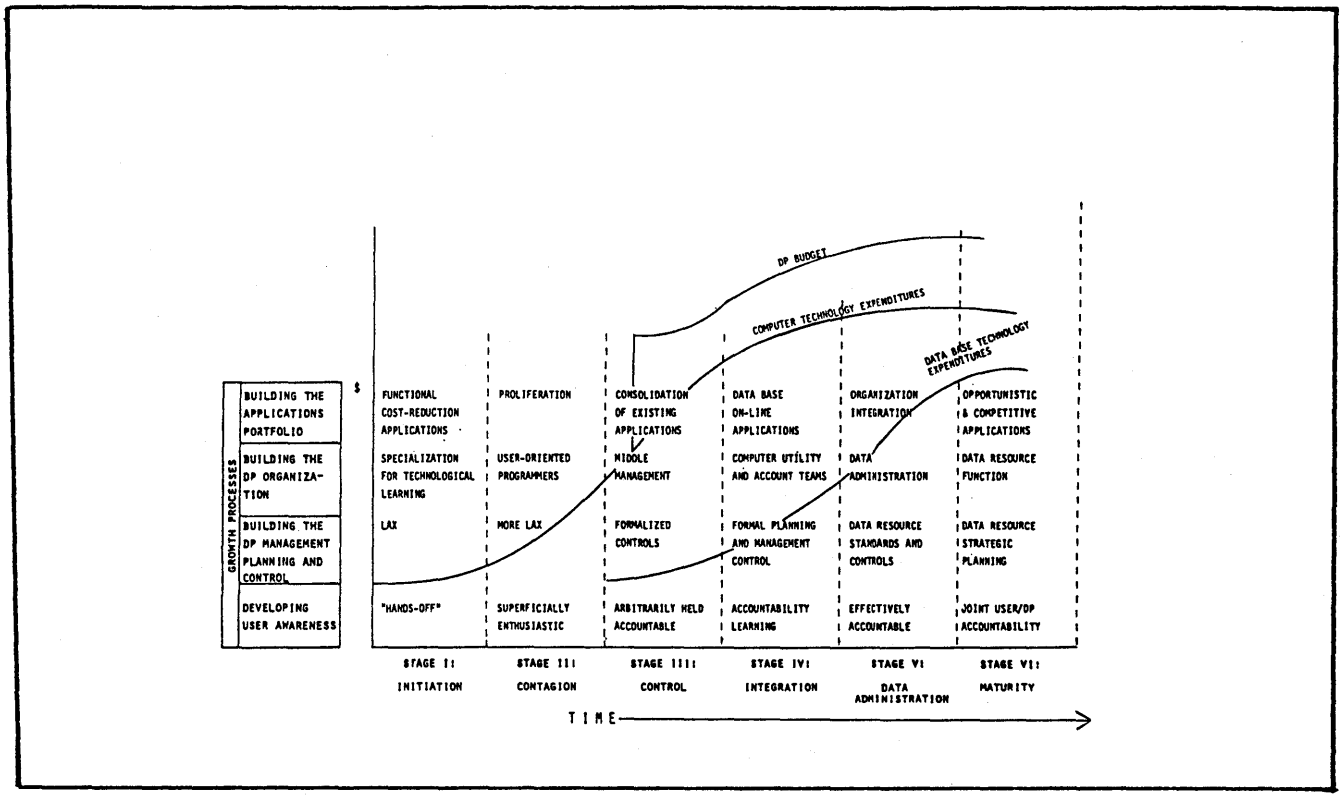


Figure 2—Six stages of data resource function

that automated their first systems in 1960. The relative permissiveness of an individual company to experiential learning is also important in understanding how a company moves through the stages.

2. *Balance Between Control and Slack*—When organizational learning is viewed as a managed process, two environments are balanced—control and slack.[2] In the control environment, all financial and performance management systems—including planning, budgeting, project management, personnel performance reviews and chargeout/cost accounting systems—are used to ensure that data processing activities are effective and efficient. In the slack environment, on the other hand, sophisticated controls are notably absent. Instead, incentives to use data processing in an experimental manner are present. The incentives come in the form of what Cyert and March call "organizational slack."[3] When management permits organizational slack in the data processing activities, they are committing more resources to data processing than are strictly necessary. The extra payment achieves another objective—nurturing innovation. When the objective is no longer present or when the benefits of overpayment are not apparent, slack is reduced. The balance between control and slack is important in analyzing the functions of each stage of organizational learning. Figure 4 shows this balance for the six stages.

In Stage I, a discretionary expenditure for data processing is made, and the objective is designated as the development of a particular application. The organization and staffing for data processing are focused on the narrow objective. Stage I is a period of primary learning about the technology and how to use it. It is also a "proving" period to demonstrate that the new technology has utility to the organization. Control is low because a high fixed-cost discretionary expenditure has been made, and there is little reason to monitor further low-level expenditures. Slack is low because the first applications are narrowly defined replacements for existing manual systems.

Once the first applications reach production status and demonstrate the utility of computer applications to the company, slack is increased to nurture widespread use of the computer, marking the transition into Stage II. Systems analysts and programmers are often recruited from user departments and trained. Formal control is kept low to promote extensive experimentation with applications in multiple functional areas. Having little experience with computer technology, management generally is unaware how fast applications proliferate. The data processing budget rises rapidly during this period because applications often are inefficient and because problems occur when attempts are made to integrate operationally-oriented applications with management's control and planning requirements.

The product of inexperienced programmers and virtually

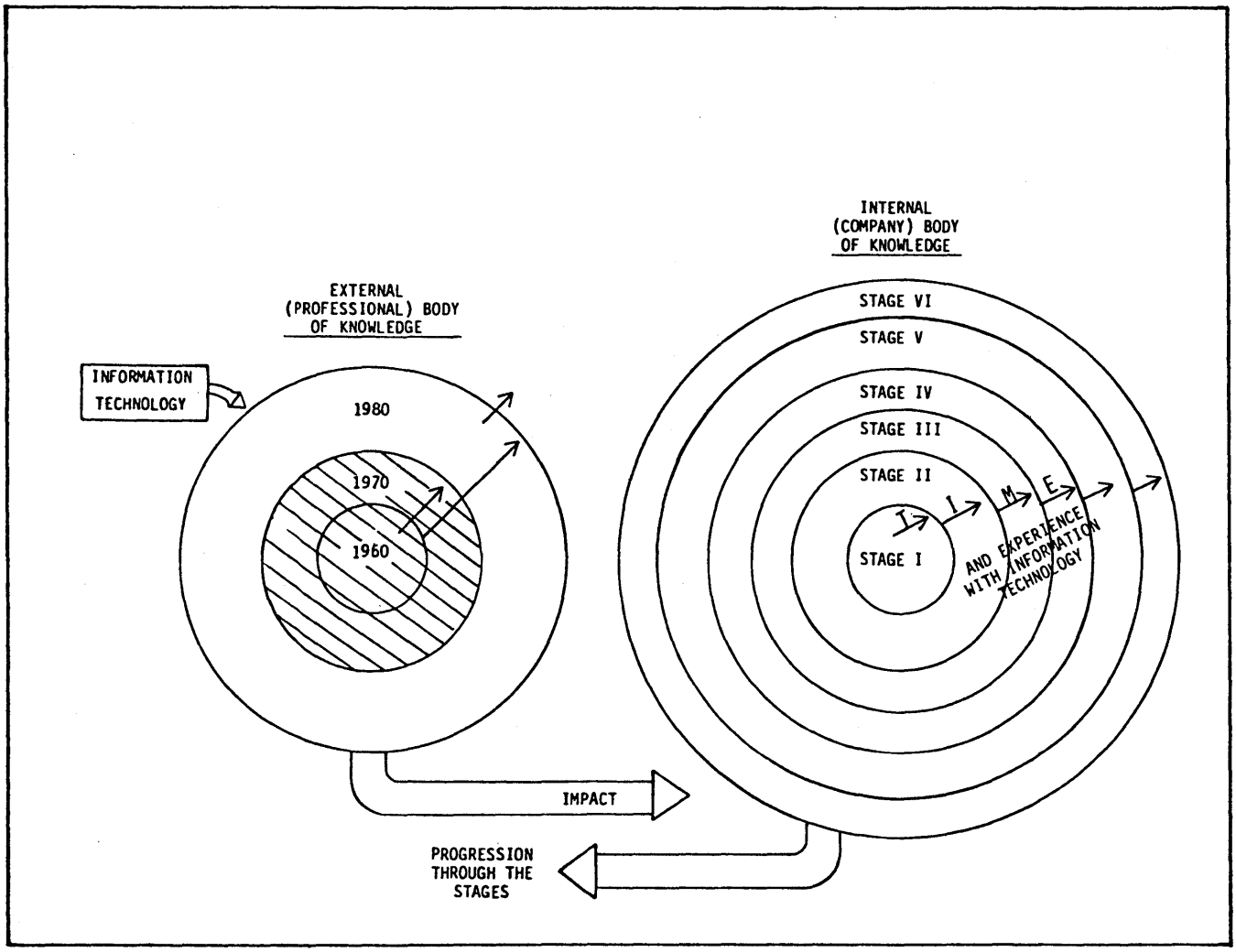


Figure 3—Development of DP bodies of knowledge and movement through stages

no standard for systems development is a portfolio of applications that are almost impossible to build upon. Thus, data processing expenses continue to rise while computer services fall off and stagnate.

When this happens, management initiates actions to increase formal control and reduce slack. The functions of Stage III are to develop appropriate management controls and to "professionalize" data processing personnel. Often management consolidates data processing activities and mandates either greatly reduced or zero budget growth. To appropriately restructure the applications portfolio, the data resource concept[4] is embraced and data-base technology is usually introduced to implement the data resource concept.

This has a confounding impact on the data processing budget as illustrated in Figure 2. Despite tight control and minimal slack in regard to computer technology, a discretionary expenditure is made for a data base application to test the viability of the technology. This is illustrated in Figure 4. Once the data processing staff is rebuilt and the data base technology is proven, the high control becomes a drawback to further progress. Low organizational slack impedes the experimentation and innovation needed to restructure the applications portfolio under the data resource concept.

In Stage IV, slack again becomes high, but control over the data processing department remains high. This is accomplished by fundamental reorganization of the data processing activities. The responsibility for applications that can be associated directly with functional users are assigned to the users and programmers and analysts are organized into Account Teams to assist users. This relieves the data processing department of the no longer logical responsibility for operating and maintaining low-level functional systems. The responsibility for multifunctional applications is vested in a user-represented steering committee. As shown in Figure 4, slack concerning data base technology is high, resulting in a rapid conversion of the existing applications portfolio to a data base structure, as well as a proliferation of integrated

STAGES	ORGANIZATIONAL SLACK		CONTROL	
	COMPUTER	DATA BASE	COMPUTER	DATA BASE
I	Low	NA (not applicable)	Low	NA
II	High	NA	Low	NA
III	Low	Low	High	Low
IV	High	High	High	Low
V	High	Low	High	High
VI	High	High	High	High

Figure 4—Balance of control and organizational slack through the six stages

applications. The architecture of these applications reflects user-orientation through the extensive use of on-line terminals in user departments.

The low control/high slack for data base technology in Stage IV leads to data processing budget growth at rates which lead to a second senior management intervention for control. Stage V is characterized by data administration and a sorting out of data processing department and user department responsibilities under the data resource concept. Slack is reduced and control is increased in order to implement the standards and policies required. The data base structure and specific technology employed are re-evaluated in light of current external and internal bodies of knowledge. Major restructuring of the data base is typically necessary in Stage V.

Once data resource management is accomplished in Stage V, slack can be increased in the specific areas of application development where it is required for innovation. The more stable and conventional data processing activities then can be subjected to a high level of control to ensure maximum efficiency. The result is a controlled data processing budget growth rate more consistent with the company's overall growth rate. These are the characteristics of Stage VI. They indicate organizational acceptance and a relative maturity in the Data Resource function.

Within the context of the two frameworks (i.e., organizational change and organizational learning about information technology), the organizational response to contemporary information technology developments can be probed.

ORGANIZATIONAL RESPONSE AND INFORMATION TECHNOLOGY DEVELOPMENTS

Among the organizational developments in companies' data processing departments that I have observed over the past five years or so, three stand out: (1) balancing centralization with decentralization, (2) data administration, and

(3) distributed processing. Each of these is associated with an initiating information technology and stage. Once initiated in a company, the organizational development or response to an underlying technology evolves through the stages. While there is a "natural" initiating stage, the organizational development may be initiated in other stages as well. Often, earlier initiation precedes a company's readiness for the organizational development and is dysfunctional to effective use and evolution of information technology. I call these stages "red flag" stages.

Organizational response #1: centralization versus decentralization

In the late 1950's and early 1960's, the initial high cost of computer processing and storage resulted in the centralized organization of most data processing departments. Then, as shown in Figure 5, the incredible cost/performance improvements in computer processing and storage and development of user-oriented programming languages gave rise to considerations for decentralizing parts of the data processing activity. Initial considerations were generally given to locating systems analysts in user departments followed by considerations to locate computers in the user departments as well.

The relative absence of planning and control of data processing activities in Stage II results in a recentralization of data processing in Stage III in order to implement the planning and control structures required to effectively manage the activity. Once the controls are implemented, the relative centralization/decentralization of the data processing activity tends to gravitate toward the overall centralization/decentralization philosophy of the company. This gravitation evolves through Stages IV, V and VI.

Stages I, II, and III are "red flag" stages because aggressive management action to effect centralization or decentralization which is contra to the natural forces of centralization in Stage I, decentralization in Stage II, and centralization in Stage III can have a disastrous impact on the data processing activity. The movement toward centralized MIS projects in the 1960's when many organizations trying such projects were in Stage II is an example of a disastrous impact. Many of these companies ended up writing off their MIS projects as total losses. The organizational learning on how to develop applications in multiple departments and manage large development had simply not developed sufficiently.

ORGANIZATIONAL RESPONSE	UNDERLYING INFORMATION TECHNOLOGY	INITIATING STAGE	RED FLAG STAGES	DEVELOPING STAGES
CENTRALIZATION VERSUS DECENTRALIZATION	PROCESSING, STORAGE, & HIGH LEVEL LANGUAGE DEVELOPMENTS	IV	I, II, III	V, VI

Figure 5—Organizational response #1: Centralization versus decentralization

ORGANIZATIONAL RESPONSE	UNDERLYING INFORMATION TECHNOLOGY	INITIATING STAGE	RED FLAG STAGES	DEVELOPING STAGES
DATA ADMINISTRATION	• DATA BASE • DATA COMMUNICATIONS	V	I, II, III, IV	VI
DP CONTROLLER		III	NONE	IV, V, VI
DP PLANNER		VI	I, II, III, IV, V	VI

Figure 6—Organizational response #2: Data administration

Organizational response #2: data administration

Data Administration receives a lot of attention in Stage III when data base technology is introduced. However, the phenomenon is not understood well enough to write appropriate job descriptions and staff it. As shown in Figure 6, the activities of data administration are usually incorporated into a DP Controller position. During Stage IV the applications portfolio is restructured using data base technology, and a critical mass of data base applications results. Data base applications, along with data communications technology, enable a user orientation.

Consequently, the experiential base that evolves by late Stage IV creates the need and appropriate organizational understanding for establishing data administration in Stage V. As the application portfolio continues to evolve with higher-level integrated applications, the opportunity for gaining competitive advantage through effective management and use of data resources emerges. This will lead to the position of Data Planner in Stage VI.

Stages I through IV are "red flag" stages for data administration because of the lack of critical experiential knowledge. However, Stages III and IV represent important opportunistic stages for laying the groundwork for effective data administration.

ORGANIZATIONAL RESPONSE	UNDERLYING INFORMATION TECHNOLOGY	INITIATING STAGE	RED FLAG STAGES	DEVELOPING STAGES
DISTRIBUTED PROCESSING	• MINI/MICRO-COMPUTER • TERMINALS	III	I, II	IV, V, VI
VICE PRESIDENT OF DATA RESOURCES		IV	I, II, III	V, VI

Figure 7—Organizational response #3: Distributed processing

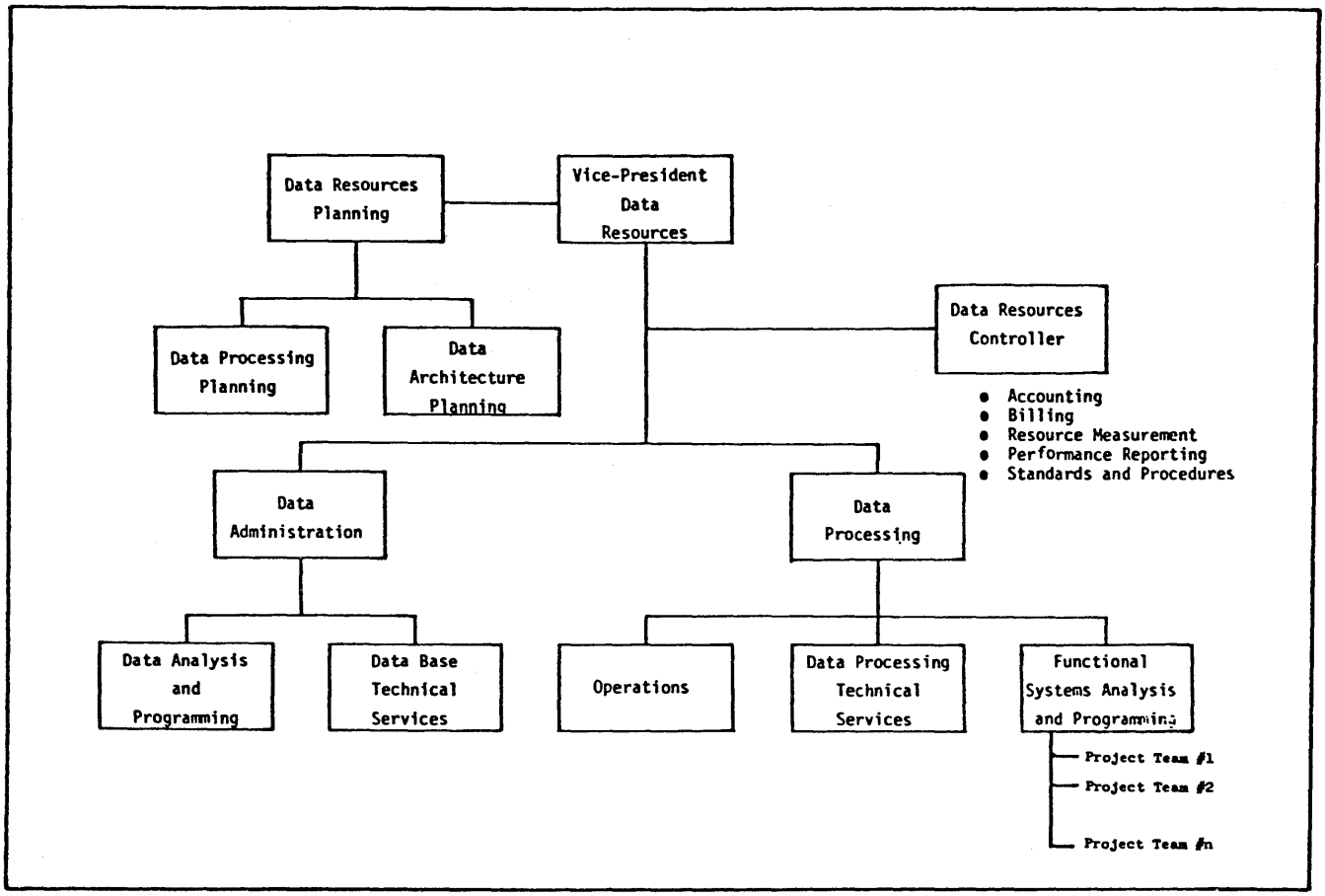


Figure 8—Stage VI data resource organization

Organizational Response #3: Distributed Processing

Distributed Processing is a direct result of technology developments in mini/micro computers, terminals, and data communications. As shown in Figure 7, distributed processing is usually initiated in Stage III. If initiated in Stages I or II, the technologies are usually overwhelming to the data processing department from both a technical standpoint and a management standpoint.

In Stage III the critical control structures are in place so as to enable a managed balancing of the relative centralization/decentralization of the data processing activity to correspond with the overall company philosophy.

The evolving complexity of distributed data processing within the context of an extensive applications portfolio that impacts every aspect of the business, leads to the need for the participation of a high-level data processing manager on the senior management team. In Stage VI, this is reflected in an organizational position such as a Vice President of Data Resources. Figure 8 illustrates the Stage VI organizational structure. Parts of the functions are distributed into the user departments dependent upon the company's overall organizational philosophy.

SUMMARY AND CONCLUSION

Organizations have been and are being remolded by developments in information technology. The advent of the computer has set off this remolding process, and the process has been quickening ever since.

The first remolding was initiated by breakthroughs in computer processing and storage technologies, and the innovation of higher level languages. The organizational response was search for the best balance between centralized and decentralized structures. The overall issue was never resolved because its resolution depended on the individual company's philosophy about centralization and decentralization.

Data base and data communications technologies set off another remolding process of organization structure. We are just beginning to witness the way that this remolding is shaping up. The six stages of the evolution of the data resource function is helpful to understanding current developments as well as future organizational developments.

Distributed processing has been set off by developments in mini-micro computer and terminal technologies. The main

effect of these technologies has been added complexity to the data resource function.

In conclusion, three points are important:

(1) Developments in information technology create organizational responses.
(2) Organizational responses of a company are dependent upon its data processing stage of maturity. The stage is a result of the organizational assimilation of external and internal bodies of knowledge. The rate of knowledge assimilation is greatly influenced by the balance of organizational slack and control.
(3) The current information technology and relative stage status of most companies has created an extremely complex data processing management environment that defies simple solutions. Organizational theories and frameworks are necessary to sort out the phenomena so that effective management action can be executed.

REFERENCES

1. Leavitt, Harold J., "Applied Organizational Change in Industry: Structural Technological and Humanistic Approaches," in James G. March, ed., *Handbook of Organizations,* Chicago, Rand McNally & Company, 1960, p. 1149.
2. Nolan, Richard L., *Management Accounting and Control of Data Processing,* New York, National Association of Accountants, 1977, pp. 22-24.
3. Cyert, Richard M. and James G. March, "Organizational Factors in the Theory of Oligopoly," *Quarterly Journal of Economics,* Vol. 70, 1956, pp. 44-64.
4. Nolan, Richard L., (Ed.), *Managing the Data Resource Function,* St. Paul, Minn., West Publishing Company, 1974, discussion of the data resource concept.

Are statistical data bases secure?*

by DOROTHY E. DENNING
Purdue University
West Lafayette, Indiana

INTRODUCTION

Statistical databases contain sensitive information about individuals. Their objective is providing access to summary statistics about groups of individuals, while denying access to the records pertaining to any particular individual. But this objective is difficult to meet, as seemingly innocuous summaries contain small vestiges of the original data. By correlating enough summaries, confidential data may be deduced and the privacy of some individual therefore compromised.

As more and more of these databases are put on-line, the problem of preventing their compromise is of growing concern. Several questions have been raised:

- How easy is it to compromise a statistical data base?
- Under what conditions are statistical data bases secure?
- Are these conditions usually met?
- Is it practical or even possible to impose these conditions on statistical data bases?

Recent studies reveal that the problem is even more difficult than at first believed. Methods once thought to significantly reduce the threat of compromise—e.g., refusing to issue summaries about small groups of individuals—are in fact easy to circumvent. This paper surveys some of these studies. We begin with a general model of a database.

DATABASE MODEL

Components

Consider a statistical database containing records of information about n individuals. Each record contains *category* and *data* fields. Categories are used to select subgroups of individuals having common characteristics; they need not be disjoint from the data values. There may also be a unique identifier field, which is neither category nor data. Statistical summaries are requested from the database with *queries* which apply to subgroups of individuals.

Table I shows a database of size n=12 containing information on customer accounts for a bank. Each individual belongs to exactly one category in each of these sets:

Sex:	M, F
Profession:	LAWYER, DOCTOR, JOURNALIST, PRESIDENT, SENATOR, BUDGET DIRECTOR
Board Member:	YES, NO
(Number of) Overdrafts:	any integer ≥0

Each individual also has data values for:

(Number of) Overdrafts:	any integer ≥0
Amount (of Overdrafts):	any integer ≥0

All examples will refer to this database.

Compromise

Compromise occurs whenever it is possible to deduce from the responses of one or more queries information not previously known about an individual. The compromise is *positive* if it reveals that the individual belongs to some category or has a particular data value. The compromise is *negative* if it reveals only that the individual does not belong to some category or have a particular data value. For example, learning that individual L had 50 overdrafts is a positive compromise; learning only that he had at least one overdraft is a negative compromise. *Partial compromise* occurs when information about at least one individual is deduced; *complete compromise* occurs when everything in the database is deduced. A database is *strongly secure* if both positive and negative compromise are impossible; it is *weakly secure* if only positive compromise is impossible.

Queries

Researchers have studied two basic forms of queries: characteristic-specified and key-specified. *Characteristic-specified queries* request statistics about all individuals in the data base satisfying a given characteristic; a *characteristic* is a logical formula using categories as operands and Boolean operators: and (·), or (+), and not (−). For a char-

* This work was supported in part by NSF Grant MCS75-21100.

acteristic C, the set of records satisfying C is called the *query set* X_C of C. For example, the characteristic C=M·(LAWYER+DOCTOR), specifying all male lawyers and doctors, has query set X_C consisting of the records for individuals A, D, E, and H. We shall use relations in the specification of characteristics, e.g., OVERDRAFTS≤2, since these are simply abbreviations for the "or" of several values, e.g., 0-OVERDRAFTS + 1-OVERDRAFTS + 2-OVERDRAFTS.

Studies of compromise of statistical data bases have considered characteristic-specified queries q(C) having one of three forms:

COUNT(C)=$|X_C|$, where $|X_C|$ is the size of X_C;

SUM(C;j)=$\sum_{i \in X_C} v_{ij}$, where v_{ij} is data field j of record i;

select(C;j)=select$_{i \in X_C}$ v_{ij}, where select is MEDIAN, SMALLEST, LARGEST, etc.

The query COUNT(C) simply returns the number of individuals satisfying characteristic C. The query SUM(C;j) returns the sum of the values in data field j for all individuals satisfying C. Note that the mean of these values can be computed from

MEAN(C;j)=SUM(C;j)/COUNT(C).

For select= MEDIAN, the query MEDIAN(C;j) returns the median value in data field j for all individuals satisfying C. Examples of queries are:

Formal Query	Answer	Informal Statement
COUNT(M·LAWYER)	3	number of male lawyers
COUNT(M·LAWYER·(OVERDRAFTS>10))	2	number of male lawyers having more than 10 overdrafts
SUM(LAWYER+DOCTOR; OVERDRAFTS)	58	total number of overdrafts of all lawyers and doctors
SUM(BUDGET DIRECTOR; AMOUNT)	$100,000	total amount of overdrafts of all budget directors
MEDIAN(LAWYER+DOCTOR; OVERDRAFTS)	1	median number of overdrafts of all lawyers and doctors

Key-specific queries request statistics for a set of k individuals identified by a list of keys I. The keys are typically the names of individuals. For a key set I, the set of records identified by the keys is the query set X_I of I. The query set size, m, is fixed for all queries. Queries for sums and selection queries are expressed as for characteristic-specified queries, with a key set I substituted for a characteristic C; e.g., SUM(I;j). Since the size of a query set is always m, queries for counts are not applicable.

Key-specified queries are of less practical interest than characteristic-specified queries, since statistical databases generally do not give out data about particular individuals. However, if each individual is identified by a unique set of categories, any key list can be expressed with characteristics. For example, the key list (A, B) can be expressed as the characteristic (M·LAWYER·$\overline{\text{BOARD MEMBER}}$+M·JOURNALIST). Thus, results which show that a database can be compromised with keys apply also to characteristics. However, caution must be taken in interpreting these results. Even though keys can be formulated as characteristics, it is not possible to do so without knowledge of the characteristics identifying the individuals named. Without this information, it is more difficult to control the composition of the query sets. For this reason, achieving compromise with characteristics may be more difficult than with keys. On the other hand, achieving compromise with characteristics may be easier than with keys since counts can be obtained for variable-size query sets.

METHODS OF COMPROMISE

Using small or large query sets

In one of the first published papers describing the problem of securing statistical databases, Hoffman and Miller described a simple algorithm for compromising a data base responding to COUNT queries.[1] Their algorithm is based on the principle of using queries which return small counts to isolate an individual. For example, consider these two queries and responses:

COUNT(M·BUDGET−DIRECTOR) =1
COUNT(M·BUDGET−DIRECTOR·(OVERDRAFTS>0))=1

If it is known that L is a male budget director, then the second query reveals that he had at least one overdraft. In general, if it is known that an individual belongs to categories $c_1, \ldots, c_k$ and if COUNT($c_1 \cdot c_2 \cdot \ldots \cdot c_k$)=1, then the query COUNT($c_1 \cdot c_2 \cdot \ldots \cdot c_k \cdot c_{k+1}$) reveals whether or not the individual also belongs to category c_{k+1} (according to whether or not the response is 1 or 0). Indeed, compromise may be possible even if the response to the first query is larger than 1, say 17, if the response to the second query is the same.

Hoffman and Millers' result is equally applicable to queries which return large counts of all but a few individuals in the data base. For example, L's overdrafts can also be determined from the queries:

COUNT($\overline{\text{M·BUDGET DIRECTOR}}$)=11
COUNT($\overline{\text{M·BUDGET DIRECTOR·(OVERDRAFTS>0)}}$) =11

These results have led researchers to consider databases which do not respond to queries q(C) when the query set size $|X_C|$ falls outside the range [k, n−k] for some k>0, where n is the size of the database. It was anticipated that k≥2 would significantly reduce the threat of compromise.

TABLE I.—Database Containing Information on Customer Accounts for a Bank

	keys	categories			data	
	Name	Sex	Profession	Board Member	Overdrafts	Amount ($)
1	A	M	LAWYER	NO	1	10
2	B	M	JOURNALIST	NO	0	0
3	C	M	PRESIDENT	NO	0	0
4	D	M	DOCTOR	NO	2	100
5	E	M	LAWYER	YES	30	50,000
6	F	F	LAWYER	NO	0	0
7	G	F	SENATOR	NO	3	50
8	H	M	LAWYER	YES	25	10,000
9	I	F	DOCTOR	NO	0	0
10	J	M	SENATOR	NO	0	0
11	K	F	JOURNALIST	NO	0	0
12	L	M	BUDGET DIRECTOR	YES	50	100,000

Trackers

Unfortunately, recent studies have shown this not to be the case. Schlörer showed that compromise may be possible even for large values of k using a "tracker" technique.[2] To illustrate his tracker, suppose k=3 for our database; i.e., no responses are given to queries which involve fewer than three or more than nine individuals. Consider these queries and responses, using as a tracker the characteristic $M \cdot \overline{\text{BUDGET DIRECTOR}}$:

$$\text{COUNT}(M \cdot \overline{\text{BUDGET DIRECTOR}}) = 7$$
$$\text{COUNT}(M \cdot \overline{\text{BUDGET DIRECTOR}} + M \cdot (\text{OVERDRAFTS}=0)) = 7$$

Because the responses to both queries are the same, it can be concluded that no male budget director had no overdrafts; therefore, L must have had an overdraft. Palme suggested a similar technique for queries that compute means.[3]

To construct a Schlörer tracker requires substantial knowledge of the characteristics identifying the individuals to be compromised. Denning, Denning, and Schwartz showed that general trackers may be found even without this knowledge.[4] Such trackers may be used to determine the answer to any COUNT or SUM query not answerable because the query set size is too small or too large. For example, if k=3, the characteristic T=LAWYER + DOCTOR is a general tracker. Let C=M·BUDGET DIRECTOR. Whereas the number of L's overdrafts cannot be determined directly from queries using only C (since $|X_c|=1$), it can be determined from queries using characteristics combining C with the tracker T as follows. First, the sum of the queries COUNT(T) and COUNT($\bar{T}$) can be used to determine the size n of the database; this gives n=12. Second, it can be deduced that L is the only male budget director from the queries COUNT(C+T) and COUNT(C+$\bar{T}$), since

$$\text{COUNT}(C) = \text{COUNT}(C+T) + \text{COUNT}(C+\bar{T}) - n$$
$$= 7 + 6 - 12$$
$$= 1$$

Third, the total number of overdrafts X can be determined by adding the responses to the queries SUM(T; OVERDRAFTS) and SUM($\bar{T}$; OVERDRAFTS), getting X=111. Finally, since L is the only budget director, the number of L's overdrafts can be determined from the queries SUM(C+T; OVERDRAFTS) and SUM(C+$\bar{T}$; OVERDRAFTS), as

$$\text{SUM}(C; \text{OVERDRAFTS}) = \text{SUM}(C+T; \text{OVERDRAFTS}) + \text{SUM}(C+\bar{T}; \text{OVERDRAFTS}) - X$$
$$= 108 + 53 - 111$$
$$= 50$$

We found further that for $k \leq n/4$ almost all data bases have general trackers (though finding them may not be easy). Trackers may also be applicable for larger values of k.

Adding dummy entries

Hoffman observed that data bases restricting the size of allowable query sets may also be subverted if records can be added to the database.[5] If query q(C) is not answerable because X_C is too small, individuals with the characteristic C are added to the data base; if X_C is too large, individuals with the characteristic $\bar{C}$ are added. This result, together with the tracker results, shows that limiting the size of allowable query sets is not an effective security measure.

Overlapping query sets

Another simple technique for compromising a data base is to construct query sets which have a high degree of overlap. For example, consider these queries and responses:

$$\text{SUM}(\text{LAWYER} + F \cdot \text{SENATOR}; \text{AMOUNT}) = \$60{,}000$$
$$\text{SUM}(\text{LAWYER}; \text{AMOUNT}) = \$60{,}010$$

Since G is the only female senator, the second query set

includes all individuals in the first query set except G. Thus, the amount of her overdrafts can be determined by subtracting the response of the second query from the first. This observation has led researchers to consider the effectiveness of methods which restrict the amount of overlap among query sets.

Linear systems

Dobkin, Jones, and Lipton considered compromising with key-specified SUM queries using query sets of size m.[6] They showed that, even if no two query sets can overlap by more than one record, compromise may be achievable in linear time (in m) without prior information, provided the data base is sufficiently large (roughly at least m^2 records). The method involves solving systems of equations for some particular data value. For example, suppose m=3 and let A_i denote the amount of overdrafts for the *ith* individual. Consider these 5 queries:

Q_1=SUM(A,B,C; AMOUNT)
Q_2=SUM(D,E,F; AMOUNT)
Q_3=SUM(A,D,G; AMOUNT)
Q_4=SUM(B,E,G; AMOUNT)
Q_5=SUM(C,F,G; AMOUNT)

$$\begin{aligned} &= A_1+A_2+A_3 && = \$\ 10 \\ &= A_4+A_5+A_6 && = \$50{,}100 \\ &= A_1+A_4+A_7 && = \$\ 160 \\ &= A_2+A_5+A_7 && = \$50{,}050 \\ &= A_3+A_6+A_7 && = \$\ 50 \end{aligned}$$

G's overdraft amount can be determined from

$$(Q_3+Q_4+Q_5-Q_1-Q_2)/3=\$50$$

Davida et al.[10] and Kam and Ullman[7] considered compromise under similar conditions.

Schwartz, Denning, and Denning extended these results to weighted summing queries.[8] We were surprised to learn that if the weights are unknown, complete compromise is possible (in linear time) provided one value in the data base is known. But compromise is impossible without initial information—even if overlap between queries is unrestricted.

Chin showed how achieving compromise with linear systems applies to characteristic-specified queries.[9] His model assumes only that no two individuals belong to the same categories.

Selection methods

Dobkin, Jones, and Lipton,[6] Davida et al.,[10] and Reiss[11] have considered key-specified queries which select some element from the query set; e.g., the median, largest, or smallest data value. Their results show that using combinatorics, compromise is generally achievable in linear time (or better) provided that no two individuals have the same data value. For example, consider these two queries involving records with distinct overdraft values:

MEDIAN(A,D,E; OVERDRAFTS)=2

MEDIAN(B,D,G; OVERDRAFTS)=2

Since D is the only individual common to both queries, the returned median value 2 must be the number of D's overdrafts. DeMillo, Dobkin, and Lipton also showed that this method of compromise applies even if the database "lies about the median value, provided it returns with some value in the query set."[12] For example, the previous example would result in a compromise of D's overdrafts even if 2 were not the true median of the query set values.

PRIVACY SAFEGUARDS

We have considered two measures—restricting the size of allowable query sets and restricting the amount of overlap between queries—which appear on the surface to reduce the threat of compromise. Yet on closer inspection, we observed that both of these measures fail to defend a system from even simple attacks. We shall now examine other privacy safeguards which will, hopefully, prove more effective.

Completely secure data bases

Several studies have reported conditions which, if imposed on the structure or contents of a database, guarantee its security.[6,7,9,13,14,15] In most cases, these conditions depend on the users having little or no supplementary knowledge about either the structure or the contents of the records. This generally rules out the users themselves (or their friends) being represented in the database. Since these conditions are seldom met, they seem to have little value in practice.

Random subfiles

Rather than making the entire Census available to statisticians, the Census Bureau distributes random subfiles of the data. The 1960 Census, for example, was distributed on tape as a $^1/_{1000}$ sample of the full Census with names and addresses removed. Also deleted was geographic information below the level of broad city-size class within the nine geographical divisions of the country.[16] As a result, the payoff from an attempted compromise of a particular individual's privacy from Census tapes is sufficiently small to not pose a serious threat. Unfortunately, this technique is applicable only to large databases; other methods are needed to prevent compromise of small databases.

Rounding schemes

Several studies have been made of rounding schemes for modifying the answers to queries.[3,16–21] One such approach is pseudo-random rounding. Truly random rounding is not

secure since the correct answer to any query can always be determined by averaging a sufficient number of responses to the same query. With pseudo-random rounding, the same query always returns the same response. This method appears to be reasonably effective, although any kind of "stochastic error" added to responses is subject to removal by methods from communication theory.

A second approach is to always round the actual response down to the nearest multiple of some integer or to report a range. For example, the database could respond to the query COUNT(LAWYER; OVERDRAFTS) with 50 (or the range 50-60) rather than the true value 56. However, as noted by Karpinski, this method is easily circumvented if records can be added to the database.[18] To determine the true value of the above query, an intruder could simply add records to the data base for fictitious lawyers having one overdraft each; after the *4th* record is added, the response becomes 60 (or the range 60-70), implying an original value of 56. Even if records cannot be added to the database, it may be possible to modify the characteristic of the query to include records of individuals for whom the number of overdrafts is known.

Partitioned databases

Another approach to preventing compromise partitions the database into groups. In the Yu and Chin scheme,[22] queries must be for characteristics involving entire groups, making it impossible to isolate a particular individual. For example, the sample database could be partitioned by profession into four groups:

Group	Identifying Characteristic	Members
1	LAWYER	A,E,F,H
2	DOCTOR	D,I
3	JOURNALIST+ BUDGET DIRECTOR	B,K,L
4	PRESIDENT+SENATOR	C,G,J

Whereas queries involving complete groups (e.g., all lawyers and doctors) are allowed; queries involving only part of a group (e.g., all male lawyers) are not. Clever query sequences can at best disclose information about an entire group. Yu and Chin show that the technique can be made effective even if the database is dynamically undergoing insertions, deletions, and updates.

The drawback is that the partitions may destroy the usefulness of the database. If the groups are not properly formed, users may not be able to obtain needed information. Whether or not a useful database can be partitioned in this way is open.

Threat monitoring

Threat monitoring techniques detect the possible occurrence of compromise. Felligi showed that it is at least theoretically possible to determine whether the response to a query, when correlated with the responses to earlier queries, could result in compromise.[23] However, the method is too cumbersome to apply in practice. Hoffman and Miller suggested that a log or audit trail of queries be kept and inspected for unusual bursts of activity or queries returning small counts.[1] Although certain compromises may go undetected, the presence of a log may provide a deterrent.

Schlörer suggested that frequency counts of categories be used to determine whether or not a given query might lead to compromise because of small counts.[21] Response is not made to any query involving categories $c_1, \ldots, c_k$ unless the product of the frequency counts $\text{COUNT}(c_i)/n$ (for $i=1, \ldots, k$) is above some threshold.

CONCLUSIONS

The "obvious" techniques for reducing the threat of compromise—e.g., limiting the range of allowable responses, restricting the amount of overlap between query sets, and certain rounding schemes—are easily circumvented. Other techniques—e.g., partitioning—may be robust, but at the price of limiting the usefulness of the data base. The conclusion is that complete privacy cannot be enforced without severely restricting the free flow of information. The questions of interest then become:

- Can we measure the relative security of a data base?
- What is an acceptable level of security?

ACKNOWLEDGMENTS

It is a pleasure to thank Mayer Schwartz and Peter Denning whose ideas and perceptions strongly influenced this paper.

REFERENCES

1. Hoffman, L. J. and W. F. Miller, "Getting a Personal Dossier from a Statistical Data Bank," *Datamation*, 16, 5, May 1970, pp. 74-75.
2. Schlörer, J., "Identification and Retrieval of Personal Records from a Statistical Data Bank," *Methods of Info. in Medicine, 14,* 1, January 1975, pp. 7-13.
3. Palme, J., "Software Security," *Datamation,* 20, 1, January 1974, pp. 51-55.
4. Denning, D. E., P. J. Denning, and M. D. Schwartz, "The Tracker: A Threat to Statistical Data Base Security," Comp. Sci. Dept., Purdue Univ., October 1977.
5. Hoffman, L. J., *Modern Methods for Computer Security and Privacy,* Prentice-Hall, 1977.
6. Dobkin, D., A. K. Jones, and R. J. Lipton, "Secure Data Bases: Protection Against User Inference," Research Report #65, Dept. of Comp. Sci., Yale Univ., April 1976.
7. Kam, J. B. and J. D. Ullman, "A Model of Statistical Data Bases and their Security," TR-207, Dept. of EECS, Princeton Univ., June 1976.
8. Schwartz, M. D., D. E. Denning, and P. J. Denning, "Securing Data Bases Under Linear Queries," *Proc. IFIPS 77,* 1977, pp. 395-398.
9. Chin, F. Y., "Security in Statistical Data Bases for Queries with Small Counts," Dept. of Comp. Sci., Univ. of Alberta, 1977.
10. Davida, B. I. et al., "Data Base Security," TR-CS-76-14, Dept. of EE and Comp. Sci., Univ. of Wis., July 1976.
11. Reiss, S. P., "Medians and Database Security," Comp. Sci. Prog., Brown Univ., October 1977.
12. DeMillo, R. A., D. Dobkin, and R. J. Lipton, "Even Data Bases that

Lie Can Be Compromised," Research Report #67, Dept. of Comp. Sci., Yale Univ., May 1976.
13. Haq, M. I., "Security in a Statistical Data Base," *Proc. Amer. Soc. Info. Sci.*, 11, 1974, pp. 33-39.
14. Haq, M. I., "Insuring Individual's Privacy from Statistical Data Base Users," *Proc. AFIPS NCC, 44,* 1975, pp. 941-946.
15. Schwartz, M. D., "Inference from Statistical Data Bases," Ph.D. Thesis, Comp. Sci. Dept., Purdue Univ., August 1977.
16. Hansen, M. H., "Insuring Confidentiality of Individual Records in Data Storage and Retrieval for Statistical Purposes," *Proc. AFIPS FJCC, 39,* 1971, pp. 579-585.
17. Fellegi, I. P. and J. L. Phillips, "Statistical Confidentiality: Some Theory and Applications to Data Dissemination," *Annals Econ. Soc'l Measurement,* 3, 2, April 1974, pp. 399-409.
18. Karpinski, R. H., "Reply to Hoffman and Shaw," *Datamation,* 16, 10, October 1970, p. 11.
19. Nargundkar, M. S. and W. Saveland, "Random-Rounding to Prevent Statistical Disclosure," *Proc. Amer. Stat. Assoc., Soc. Stat. Sec.,* 1972, pp. 382-385.
20. Reed, I. S., "Information Theory and Privacy in Data Banks," *Proc. AFIPS,* 42, 1973, pp. 581-587.
21. Schlörer, J., "Confidentiality of Statistical Records: a Threat Monitoring Scheme for On-Line Dialogue," *Methods of Info. in Medicine,* 15, 1, January 1976, pp. 36-42.
22. Yu, C. T. and F. Y. Chin, "A Study on the Protection of Statistical Data Bases," Dept. of Comp. Sci., Univ. of Alberta, 1977.
23. Fellegi, I. P., "On the Question of Statistical Confidentiality," *J. Amer. Stat. Assoc., 67,* 337, March 1972, pp. 7-18.

SECURATE—Security evaluation and analysis using fuzzy metrics*

LANCE J. HOFFMAN
The George Washington University
Washington, DC

and

ERIC H. MICHELMAN
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

DON CLEMENTS
Teknekron, Inc.
Berkeley, California

INTRODUCTION

In the last several years, methods for controlling security in computer systems have become widely known. While in 1972 there were only one or two books and three bibliographies on computer security and privacy, in 1977 there were at least 15 books and six bibliographies on the topic. Expanding public concern with the problem is evidenced by a great deal of federal, state and local legislation.[1,2] Governmental agencies such as the National Bureau of Standards, the Defense Department, and the National Science Foundation are all sponsoring research efforts in the area, as have some private manufacturers.[3–5]

One consequence of this work is that a significant part of the computing community is now aware of techniques[6] for maintaining security in computer systems. Unfortunately, the question of how to measure the costs and effectiveness of the various security methods is still largely unexplored. Only recently has there been any reliable work done on costs of privacy transformations or authentication methods, and researchers have only scratched the surface in investigating metrics for security systems.[7] While some more formal work has begun,[8–13] there has so far been very little useful application of this theoretical work to practical security decisions. One exception is a privacy cost model which has been recently introduced.[14] This work assesses the impact of various privacy regulations upon data processing installations which handle personal information. Goldstein discusses conversion costs to meet privacy requirements and the "privacy increment" to installation operational costs. He does not, however, deal directly with the effectiveness of security techniques.

We will address the question of security effectiveness. We here describe SECURATE, a computer installation security evaluation and analysis system, which is based upon a model of a computer installation as a set of triples composed of objects, threats, and security features. SECURATE uses security rating functions based on fuzzy set theory to provide security ratings for an installation as a whole as well as for subsections. It helps to determine weak and strong points and facilitates the comparison of alternative security designs. SECURATE is not meant to be a substitute for a human decision maker.

The SECURATE user first supplies a set of object-threat-feature triples necessary to describe a computer system from a security point of view. OBJECTS are defined as the resources within a computing system, the loss of which would have a cost to the owner. THREATS are activities which a potential intruder may employ to gain unauthorized access to an object. This term also refers to chance events which may jeopardize an object. FEATURES are protective measures which present some degree of resistance to a threat. The second section describes this framework in more detail.

After the system description is complete, the user is asked to specify his or her outlook on security; some view total system security as an all-or-nothing thing, while others see it as dependent in various ways on a system's individual components. Based upon the user's outlook and the system description, SECURATE supplies evaluation functions which return natural-language security ratings of the system or its subsystems. The fifth section describes these evaluation functions.

After the system was implemented in APL, it was used at

* Work supported in part by Grant MCS76-09214 from the National Science Foundation and by the University Committee on Research, The George Washington University.

seven installations by students who were doing risk analyses of the installations. Feedback from this initial group of users is discussed in the seventh section.

TECHNICAL BASIS

The technical basis for SECURATE is in earlier work[15] which defines a security system as a set of objects, each with a loss value; a set of threats, each with a likelihood; and a set of features, each with a resistance. This work addresses the problem of imprecision in the approximation of values, likelihoods, and resistances by using linguistic variables[16] to combine the values of these variables into security ratings.

Clements' model groups resources within computing systems which are vulnerable to some security threat as the set O of security objects. Each object in the set possesses a loss value to its owner.

Associated with each security object O_j is a number of potential intrusion activities; all of these together form the set T of security threats. Each threat T_i has associated with it a likelihood of occurrence.

The object-threat relations form a bipartite directed graph (Figure 1) in which edge T_iO_j exists only if threat T_i is a viable means of compromising object O_j. The relations of threats to objects is not one to one; a threat may compromise any number of objects and an object may be vulnerable to more than one threat.

Finally, there is the set F of security features. A security feature performs a "firewall function" by presenting some degree of resistance to a penetration attempt. The set of security features transforms the bipartite graph of Figure 1 into the tripartite graph of Figure 2. In a "protected" system all edges are of the form T_iF_k and F_kO_j. Any edge of the form T_iO_j identifies an unprotected object. Each triplet (T_i, F_k, O_j) in $T\times F\times O$ contributes to the overall security rating of a system, as we shall see below.

In attempting to specify the object values, threat likelihoods, and feature resistances, one is confronted with the problem of imprecision. In evaluating a computer system's security, one generally relies on imprecise human judgment to provide approximate measures of the effectiveness of security features. The problem is aggravated when one attempts to produce security ratings for entire systems from these measures. The assignment of a numerical security rating is inconsistent with the complexity of data processing installations and the imprecision of the underlying measures. Stating that an installation is ".65 secure" has limited appeal for imparting a sense of how secure an installation is. In addition, the precision implied by such a rating should be viewed skeptically.

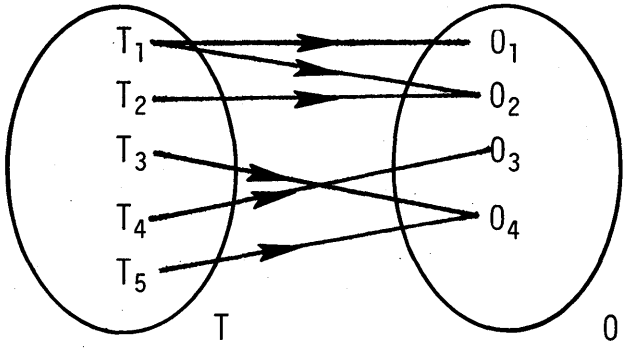


Figure 1—The threat-object relation

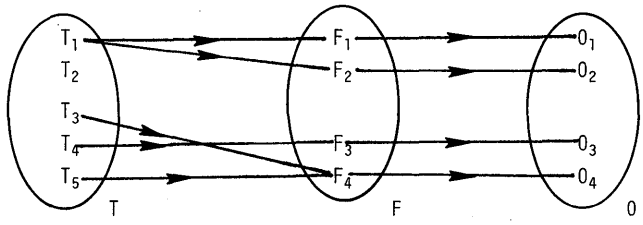


Figure 2—The basic security system

Clements suggested that it is possible to make meaningful measurements of the security of a computer system through the use of linguistic variables—variables which assume values which are words rather than numbers. Using this approach, the object values, threat likelihoods, and feature resistances (as well as the resultant security rating) are specified in terms such as *high, low,* and *medium.* Appropriate modifiers provide finer resolution by allowing terms such as *very high, somewhat low,* etc.

The user assigns linguistic values (*high, medium, very high,* etc.) to the component variables (loss value, likelihood, resistance) for each (object, threat, feature) triplet in the system. These measures determine the contribution of that particular triplet to total system security. How this is done is discussed in a later section.

Each linguistic variable is a fuzzy set whose members are real numbers in the interval [0,1]. These values comprise the *compatibility function,* μ_f, for the specific linguistic variable. For example, if $\mu_{\text{high}}(0.8)=0.9$, 0.9 represents the degree to which a non-fuzzy rating of 0.8 on a *base scale* agrees with a fuzzy rating of *high.* Figure 3 illustrates what the complete compatibility functions for typical *high* and *very high* terms might be. More detail on linguistic variables, base scales, and compatibility functions can be found in Reference 16.

OBJECT HIERARCHY AND LINGUISTIC TERMS

SECURATE provides a "default" hierarchical structure of objects commonly found in computer installations.[17] A small part of this object hierarchy and the corresponding threats and features are presented in the appendix.

There are two phases involved in using the system: (1) typing in a description of the installation and (2) using the security analysis functions.

Object value, threat likelihood, and feature resistance for each security point of interest are first specified by the user in terms of linguistic variables. The vocabulary and Backus-

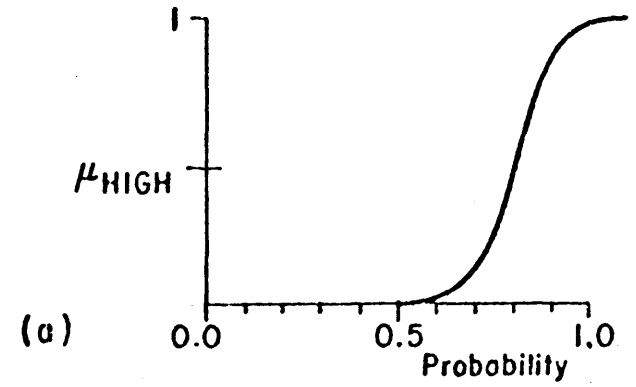


Figure 3a—Typical compatibility function of "high" probability

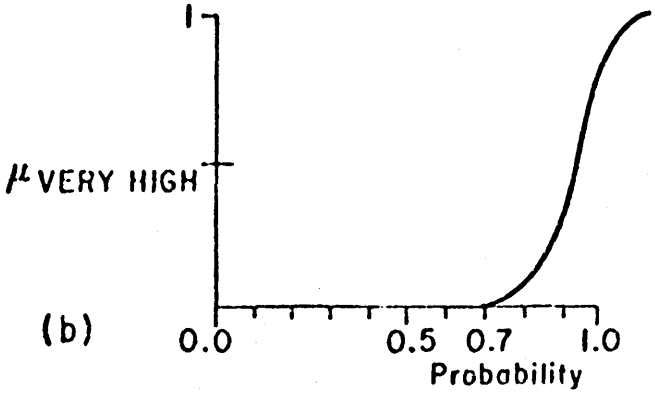


Figure 3b—Typical compatibility function of "very high" probability

Naur form syntax of the rating language, along with examples, are shown in Figure 4.

Once the installation to be analyzed is described in terms of these triples, the functions described in the fifth section are invoked by the user to evaluate and analyze its security.

THE USER INTERFACE

From the start of the SECURATE project, an important objective was to design and implement the system so that it would be as hospitable to the users as possible. Our goals concerning user-oriented features were primarily to keep the system simple, easy to use, and non-tedious. More specifically, we were concerned with the following points:

(a) User Understanding—for obvious reasons, achieving adequate user understanding is very important. Not only won't the system be useful if the user doesn't understand it, but it won't be used.

(b) Simple, Non-tedious Interface—a similar, much simpler system had been developed earlier by a student as a term project. A unanimous criticism of that system was that it took too long to use and the data entry was too tedious. As our system was to require considerably more information, it seemed important to keep the interaction as short, concise, and painless as possible.

(c) Useful Evaluation Functions—while it may seem that this is the most important point, it may actually be the least. A system which a user understands and is comfortable using is more likely to be used and be helpful than a system that doesn't possess these qualities, even if the evaluation functions provided by the first aren't quite as useful as those provided by the second.

Since the object hierarchy was used as a guide for collecting user data, it provided a convenient basis for structuring the input. We drew up forms which corresponded in format exactly to what appeared on the screen. The user wrote down on the forms only the necessary information and then transferred it easily to the system. These forms, by integrating the system commands with the input data in a coherent way, familiarized the users with the system's operation prior to their using it. Figures 5a and 5b show an example of the input form and the corresponding data entry. Further information concerning the user interface is given in Reference 18.

SECURATE assumes that the installation will be similar to that modeled by the hierarchy in Appendix A, although the user can modify the hierarchy appropriately as he or she supplies the triples information. SECURATE leads the user through the hierarchy (and any user modifications to it), providing the opportunity at each node to add offspring or specify triples. If a triple is specified for an object with offspring, it is assumed to refer to that object and each of its offspring. Objects, threats, and features appearing in more than one triple may have different values, likelihoods or resistances, respectively.

⟨sentence⟩ ::= ⟨compound phrase⟩ | ⟨simple phrase⟩
⟨compound phrase⟩ ::= ⟨conjunctive phrase⟩ | ⟨range phrase⟩
⟨simple phrase⟩ ::= ⟨relational phrase⟩ | ⟨hedged primary⟩
⟨conjunctive phrase⟩ ::= ⟨relational phrase⟩ AND ⟨relational phrase⟩
⟨range phrase⟩ ::= ⟨hedged primary⟩ TO ⟨hedged primary⟩
⟨relational phrase⟩ ::= ⟨composite relation⟩ THAN ⟨hedged primary⟩
⟨composite relation⟩ ::= ⟨relation hedge⟩ ⟨relation⟩ | ⟨relation⟩
⟨relation hedge⟩ ::= NOT | MUCH | SLIGHTLY
⟨relation⟩ ::= LOWER | HIGHER
⟨hedged primary⟩ ::= ⟨hedge⟩ ⟨primary⟩ | ⟨primary⟩ | ⟨fuzzy number⟩
⟨hedge⟩ ::= NOT | VERY | MOREORLESS | QUITE ⌈ PRETTY | SORTOF | REALLY | EXTREMELY | INDEED
⟨primary⟩ ::= LOW | HIGH | MEDIUM
⟨fuzzy number⟩ ::= ⟨fuzzifier⟩ ⟨number⟩
⟨fuzzifier⟩ ::= ABOUT
⟨number⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

Figure 4a—BNF grammar of rating language

high
low
medium
not high
moreorless medium
indeed low
low to medium
(about 4) to about 6
slightly lower than pretty high
not higher than medium
(much higher than low) and slightly lower than sortof high

Figure 4b—Examples of terms available in rating language

OBJECT NO:				1
	ADD, A name or number			A METERING EQUIPMENT
	VALUE, V object value			
THREAT NO	THREAT LIKELIHOOD	FEATURE NOS		FEATURE RESISTANCE

OBJECT NO:				11
	ADD, A name or number			
	VALUE, V object value			V VERY HIGH
THREAT NO	THREAT LIKELIHOOD	FEATURE NOS		FEATURE RESISTANCE
8	MEDIUM	2		PRETTY HIGH
10	PRETTY LOW	.29	30	MEDIUM

Figure 5a—Data input form

ENTER THE OBJECT NUMBER FOR THE NEXT OBJECT:
1
HARDWARE
:
ADD METERING EQUIPMENT
METERING EQUIPMENT RECEIVED OBJECT NUMBER 71
:
O
OBJECT NO 11, *CENTRAL MACHINE IS NEXT*.
:
V VERY HIGH
THREAT NO THREAT LIKELIHOOD FEATURE NOS FEATURE RESISTANCE
→
8 *MEDIUM* 2 *PRETTY HIGH*
→
10 *PRETTY LOW* 29 30 *MEDIUM*
→

:

Figure 5b—Data entry information typed by the system is underlined

Once the triples are entered, they may be printed out using the DISPLAY function. For each triple this prints out: the triple number; the object name, number, and value; the threat name, number, and value; and the feature resistance. Figure 6 shows the DISPLAY output for a sample system. The information describing the installation is automatically saved and may be used later with repeated applications of the system.

EVALUATION FUNCTIONS

The general system structure is shown in Figure 7. There are several security evaluation functions implemented. Each produces an evaluation based upon the *section* of the system being considered *and* upon a given *outlook* on security.

The possible sections of a system which can be evaluated are the following:

(a) Overall System—This function returns a security rating for the entire installation. That is, it rates the entire set of triples.
(b) All subsections of a Section—with either the top level of the installation hierarchy or one of the subsections specified, this function returns an individual rating for each subsection at the next lower level. For example, if in our sample system, the top level of the hierarchy was specified for a sectional analysis, security ratings would be printed out for each of the following subsections: hardware, software, and the computer center (Figure 8).
(c) Individual Subsection—a security rating is returned for a specified subsection of the installation. Only triples for that subsection (including offspring) are considered (see Figure 9).

In choosing an outlook, the user in effect describes how he or she views security. Once an outlook is chosen, it stays in effect for all of the evaluation functions until it is respecified.

The *outlooks* are the following:

(a) Weakest Link—this will look for the weakest feature resistance and return that as the security rating. The theory here is that the system is only as secure as its weakest link (Figure 10).
(b) Selected Weakest Link—this produces a weakest link rating based on those triples which satisfy the condi-

DISPLAY

FOLLOWING IS A LIST OF OBJECTS ADDED, THEIR ASSIGNED OBJECT NUMBERS, AND THEIR PARENT IN THE HIERARCHY:

OBJECT	*OBJECT NO*	*PARENT*
METERING EQUIPMENT	71	1

	OBJECTS	*THREATS*	*FEATURES*
TRIPLE NO	★ *NUMBER NAME* ★ *VALUE*	★ *NUMBER NAME* ★ *LIKELIHOOD*	★ *NUMBER NAME* ★ *RESISTANCE*
1	★ 11 *CENTRAL MACHINE* ★ *VERY HIGH*	★ 8 *UNAUTHORIZED USE* ★ *MEDIUM*	★ 2 *GUARD* ★ *PRETTY HIGH*
2	★ 11 *CENTRAL MACHINE* ★ ★ *VERY HIGH*	★ 10 *HUMAN ERROR* ★ ★ *PRETTY LOW*	★ 29 *OPERATOR TRAINING* ★ 30 *DETAILLED, ACCURATE, ACCESSIBL* ★ *MEDIUM*
3	★ 12 *STORAGE MEDIA* ★ ★ *HIGH*	★ 13 *UNAUTHORIZED READ* ★ ★ *HIGH*	★ 43 *DATA ENCRYPTION* ★ 44 *EFFECTIVE STORAGE ACCESS CONTR* ★ *PRETTY LOW*
4	★ 12 *STORAGE MEDIA* ★ *HIGH*	★ 11 *THEFT* ★ *LOW*	★ 31 *PHYSICAL ACCESS CONTROLS* ★ *FAIRLY HIGH*
5	★ 71 *METERING EQUIPMENT* ★ *LOW*	★ 4 *HARDWARE TAMPERING—MODIFIED* ★ *LOW*	★ 21 *LOCKS AND ALARMS ON MACHINE CO* ★ *HIGH*
6	★ 22 *PROGRAMS* ★ *MEDIUM*	★ 46 *INADEQUATE DEBUGGING* ★ *FAIRLY HIGH*	★ 114 *PROGRAM TESTING AND VALIDATION* ★ *(FAIRLY LOW) TO MEDIUM*
7	★ 23 *DATA* ★ ★ *HIGH*	★ 20 *UNSECURED STORAGE MEDIA* ★ ★ *HIGH*	★ 60 *ADEQUATE AND ENFORCED LIBRARY* ★ 61 *USAGE LOG* ★ *PRETTY LOW*
8	★ 23 *DATA* ★ ★ *HIGH*	★ 33 *EXPOSED OUTPUT* ★ ★ *MEDIUM TO HIGH*	★ 90 *CLEAN DESK POLICY* ★ 91 *USER EDUCATION* ★ *LOW*
9	★ 23 *DATA* ★ ★ ★ *HIGH*	★ 43 *DATA PREPARATION ERRORS* ★ ★ ★ *PRETTY HIGH*	★ 103 *SECOND PERSON VERIFICATION* ★ 104 *CHECKSUMS* ★ 105 *SOFTWARE CHECKS* ★ *HIGH*

Figure 6—Display of Triples of the Sample System

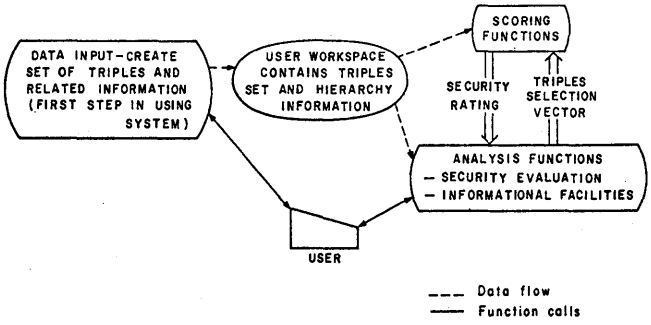


Figure 7—System structure

tion that either the object value or the threat likelihood is greater than a user specified minimum. The theory here is that one would only want to consider triples where the object is of at least a certain value or the threat is of at least a certain likelihood (see Figure 8).

(c) Fuzzy Mean—this performs a fuzzy mean[15] on the feature resistances and returns the result as the rating. The theory here is that a system's security is the mean of the security of its components (Figure 11).

(d) Weighted Fuzzy Mean—this performs a fuzzy mean on the feature resistance weighted by the greater of the object value and threat likelihood for each triple. The theory is that of (c), with the additional assumption that the more valuable objects and those with more likely threats should receive greater weight in the security rating (see Figure 12).

(e) Fuzzy Mean with Each Major Subsection Weighted

```
ENTER THE NUMBER OF THE RATING FUNCTION YOU WISH TO USE: 2
      SECTIONALRATING
ENTER THE PARENT OBJECT NUMBER (0 FOR THE TOP LEVEL IN THE HIERARCHY): 0
SPECIFY MINIMUM FOR HARDWARE : MEDIUM
4 ELEMENT(S) USED
SPECIFY MINIMUM FOR SOFTWARE : HIGH
1 ELEMENT(S) USED
SPECIFY MINIMUM FOR THE COMPUTER CENTER : PRETTY HIGH
4 ELEMENT(S) USED

**********************************************************************************************
*
* NAME                    RATING (USING SELECTED WEAKEST LINK)
*
* HARDWARE                PRETTY LOW
* SOFTWARE                PRETTY HIGH
* THE COMPUTER CENTER     PRETTY HIGH
*
**********************************************************************************************
```

Figure 8—Sectional Ratings for Sample System

```
      SETRATE 3
      INDIVIDUALRATING
ENTER THE NUMBER OF THE OBJECT TO BE RATED: 2

**********************************************************************************************
*
* NAME                    RATING (USING FUZZY MEAN)
*
* SOFTWARE                SORTOF MEDIUM
*
**********************************************************************************************
```

Figure 9—Individual Subsection Rating (Software Subsection) of Sample System

```
ENTER THE NUMBER OF THE RATING FUNCTION YOU WISH TO USE: 1
      OVERALLRATING

**********************************************************************************************
*
* NAME                    RATING (USING WEAKEST LINK)
*
* THE INSTALLATION        LOW
*
**********************************************************************************************
```

Figure 10—Rating Using Weakest Link Outlook and Overall Section of Sample System

```
      SECTIONALRATING
ENTER THE PARENT OBJECT NUMBER (0 FOR THE TOP LEVEL IN THE HIERARCHY): 0

**********************************************************************************************
*
* NAME                    RATING (USING FUZZY MEAN)
*
* HARDWARE                ((SLIGHTLY LOWER) THAN FAIRLY HIGH) AND (SLIGHTLY HIGHER) THAN SORTOF HIGH
* SOFTWARE                SORTOF MEDIUM
*
**********************************************************************************************
```

Figure 11—Partial Rating of Overall Section of Sample System Using Fuzzy Mean Outlook

```
**********************************************************************************************
*
* NAME                    RATING (USING FUZZY MEAN WEIGHTED BY VALUE)
*
* OPERATING SYSTEM        (MOREORLESS MEDIUM) TO (SORTOF HIGH)
* PROGRAMS                MOREORLESS MEDIUM
* DATA                    SORTOF MEDIUM
*
* THE LOWEST RATING WAS GIVEN TO:
*     DATA
*
**********************************************************************************************
```

Figure 12—Sample System Software Sectional Rating Using Fuzzy Mean Weighted by Value Outlook

by Maximum Object Value—for each major subsection of the object specified, this finds the fuzzy mean of the resistances. It then weights these fuzzy means by the maximum object value found in the triples for each major subsection and averages these weighted means. The theory is similar to (d), but with the assumption that the major subsections should be weighted by their relative values, irrespective of the number of triples they each have.

THE USE OF APL

APL is extremely well suited to applications involving linguistic variables and fuzzy set operations. Using appropriately named functions and variables, the linguistic variables can be easily converted into the corresponding base variables using the APL "execute" function. For example, HIGH might be a vector of (0 0 0 0 0 .1 .5 .9 1), representing the linguistic variable *high*. VERY might be a function which sharpens the curve given to it as its argument, perhaps squaring the argument. Then as shown in Figure 13, if VALUE were a variable containing the character string "VERY HIGH", executing it would return the vector (0 0 0 0 0 .01 .25 .81 1), representing the base variable for the linguistic variable *very high* (Figure 3 gives the curves representing *high* and *very high*). The important point here is that APL eliminates the need to do any parsing of the input values; the linguistic variables input are stored in character string format and used as input to the scoring functions at the appropriate time. Additionally, the built-in APL matrix operations are well suited to the fuzzy set operators, which use vectors and matrices extensively. These operators are described in detail in Reference 15.

```
      ∇VERY[□]∇
      ∇ OUT←VERY IN
[1]   OUT←IN×IN
      ∇

      HIGH
 0 0 0 0 0 0.1 0.5 0.9 1

      VALUE
VERY HIGH

      ⍎VALUE
 0 0 0 0 0 0.01 0.25 0.81 1
```

Figure 13—APL Execution of Linguistic Variables

On the negative side, APL is interpretive; this makes it significantly slower than compiled programs for repeated runs. In addition, it is poorly suited to applications not involving vectors or arrays. The latter point is important for the security evaluation system since most of the code deals with the user interface and the analysis functions. Not only were these awkward to program, but they run rather slowly (these two points not being unrelated). The rating functions, however, which make heavy use of the matrix capabilities while performing fuzzy set operations, are well suited to APL.

USER REACTIONS

Shortly after development started, we arranged to have the system (running on an IBM 370/145 system under the CP/CMS operating system) tested by students who were doing risk analyses of computer installations as term projects. SECURATE was used to analyze seven installations, including one at a large bank and one at a large utility company.

From written evaluations solicited from each of the users, as well as from conversations with them, it seems clear that the system achieved its goal of increasing understanding of installation security. In fact, a couple of users remarked that just filling out the forms made the strengths and weaknesses of an installation's security a lot clearer. Apparently focusing their thoughts into a logical, well-defined framework enabled them to view the situation more clearly and—even before using the system—to gain some of the insights we had hoped the system would provide.

The most interesting observations were those concerning the use of fuzzy variables. There appears to be definite tradeoff between user acceptance and ease of use. The concept of fuzzy variables was new to all of the users and was greeted with a certain amount of skepticism. While their acceptance of the idea grew as they continued to be exposed to it and had experience in using it, some of them remained skeptical. On the other hand, some of them commented, and we strongly feel this to be true, that the use of these words instead of numbers was a definite help in minimizing the

tedium involved in collecting the input data. The largest installation turned out to be represented by 136 triples, which cam to over 300 different measurements the user had to make. Pinpointing each one on a scale of 1 to 10 appears to us to be a lot more taxing than rating each one as a linguistic variable. Although we didn't do any comparative studies (which in retrospect would have been a good idea), many users seemed to agree with this in informal discussions.

SUMMARY

We have described an interactive security evaluation system which uses fuzzy metrics. The system models the installation to be analyzed as a set of object-threat-feature triples. The associated measures—object values, threat likelihoods, and feature resistances—are then used as input to security evaluation functions. The user specifies these features in terms of "fuzzy" linguistic variables. The system, implemented in APL, is currently operational on an Amdahl 470 computer.

REFERENCES

1. Computer and Business Equipment Manufacturers Association, Periodical Lists of State Legislation on Privacy and Security.
2. Public Law 93-579, "Privacy Act of 1974," 93rd Congress, December 31, 1974.
3. "What Every Executive Should Know About Privacy in Information Systems," Project SAFE, State of Illinois, 1974.
4. Saltzer, J. H., "Ongoing Research and Development on Information Protection," *ACM Operating Systems Review*, Vol. 8, No. 3, July 1974.
5. Goldstein, R. C., *The Cost of Privacy*, Honeywell Information Systems, 40 Guest Street, Brighton, MA., 02135.
6. Hoffman, Lance J., *Modern Methods for Computer Security and Privacy*, Prentice-Hall, Englewood Cliffs, N.J.
7. Hoffman, Lance J., "Constructing Security Ratings for Computer Systems," *Proceedings of the 1974 IEEE National Telecommunications Conference*, San Diego, California.
8. Bell, D. and L. J. LaPadula, "Secure Computer Systems: A Mathematical Model," MITRE Corporation, Bedford, MA., MTR-2547, Vol. II, Nov. 1973, EST-TR-73-278.
9. Andrews, Gregory R., "Partitions and Principles for Secure Operating Systems," *AFIPS Conference Proceedings*, Vol. 44, 1975, pp. 177-180.
10. Popek, G. J., and C. S. Kline, "Verifiable Secure Operating System Software," *AFIPS Conference Proceedings*, Vol. 43, 1974.
11. Walter, K. G., W. F. Ogde and W. C. Rounds, et al., "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 23, 1974.
12. Harrison, M. A., W. L. Ruzzo and J. D. Ullman, "Protection in Operating Systems," *Communications of the ACM*, Vol. 19, No. 8, August 1976, pp. 461-471.
13. Hartson, J. R. and D. K. Hsiao, "A Semantic Model for Database Protection Languages," *Proceedings Second Very Large Data Base Conference*, Brussels, Belgium, 1976.
14. Goldstein, R. C., H. H. Seward and R. L. Nolan, "A Methodology for Evaluating Alternative Technical and Information Management Approaches to Privacy Requirements," NBS Technical Note 906, National Bureau of Standards, June 1976.
15. Clements, D., "Fuzzy Ratings for Computer Security Evaluation," Memorandum No. UCB/ERL M77/41, June 1977, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.
16. Zadeh, L. A., "The Concept of a Linguistic Variable and its Application to Approximate Reasoning," Part 1, *Information Science*, Vol. 8, 1975, pp. 199-249; Part II, *Information Science 8*, 1975, pp. 301-357; Part III, *Information Science*, 9, 1975, pp. 43-80.
17. Michelman, Eric H., "A Practical Framework for Computer Installation Security," Memorandum No. M77/4, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.
18. Hoffman, Lance J., Eric H. Michelman and Don Clements, "SECURATE User's Manual," Memorandum No. UCB/ERL M77/49, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.
19. Parker, Donn B., *Crime by Computer*, Charles Scribner's Sons, New York, 1976.

APPENDIX—THE OBJECT HIERARCHY, THREATS, AND FEATURES

The structure of the *threats* is based on the *object hierarchy*, which is used as an outline. *Threats* are listed after the *objects* they refer to, the objects being specified by name and number from the object hierarchy. A threat listed after a non-terminal node of the object hierarchy refers to all objects descending from the node. Features which counter threats follow the listing of the threats.

Partial object hierarchy

2. Software
   - 2.1 Operating system
   - 2.2 Programs
     - 2.2.1 Applications
       - 2.2.1.1 Source
       - 2.2.1.2 Non-source
     - 2.2.2 Contract programs and packages
     - 2.2.3 System utilities
     - 2.2.4 Test programs
   - 2.3 Data
     - 2.3.1 Personal data
       - 2.3.1.1 Payroll
       - 2.3.1.2 Personnel
       - 2.3.1.3 Other personal data (Privacy Act of 1974, §3(a)(4))
     - 2.3.2 Institution data
       - 2.3.2.1 Marketing
       - 2.3.2.2 Financial
       - 2.3.2.3 Operations
       - 2.3.2.4 Planning
       - 2.3.2.5 Other

Partial Threats Listing

2. Software

16 A. Unauthorized access: R/W/E

17 Modification of operating system and system routines

18 Inadequate controls on I/O facilities

19 Password compromise

20 Unsecured storage medium

21 Access outside of allocated memory

22 Modification of stored state vector

23 Unauthorized CE activity
24 Line tapping and spoofing
25 Erroneous or inadequate usage of protection facilities
26 B. Unauthorized access: read
27 Extra copies of output printed
28 duplicates printed
29 printing restarted before end
30 Use of erroneous distribution labels
31 Use of erroneous distribution lists
32 Theft of mail
33 Exposed output
34 in user possession
35 within distribution system
36 at operator's console
37 work in progress
38 Unauthorized reading of terminal buffers
39 Indirect exposure of output
40 C. Unauthorized access: write
41 Modification or spoof of mail transactions
42 Unauthorized modification of data during preparation
43 Data preparation errors
44 Modification of original written data input
2.1 Operating system
45 Defective implementation
2.2 Programs
46 Inadequate debugging
47 Incomplete operation specifications
48 Inadequate or erroneous error handling
49 Exposure following abnormal end
50 Improper operation
2.2.2 Contract programs and packages
51 Dishonest programs
2.2.4 Test programs
52 Unexpected alteration of real data

Features

Feature No.	Threat Numbers	Feature Name
46	16	Effective authorization and access control mechanism
47		Minimum authorization policy
48	17	Effective authorization and access control mechanism
49		Minimum authorization policy
50		Dual authorization required for changes
51		Super user authorization required for changes
52		Log of attempted violations
53	18	Self-modifying I/O routines not allowed
54	19	Direction in password choice
55		Store in encrypted form
56		Automatic delay after invalid login attempt
57		Encrypted transmissions to terminals
58		Use of interactive authentication procedure
59	20	Adequate access controls
60		Adequate and enforced library facility
61		Usage log
62		Proper labelling
63	21	Proper system design
64		Effective authorization and access control mechanism
65		Adequate I/O controls
66		Protection of state vector
67	22	Storage in protected storage
68	23	Administrative controls
69		Human verification
70		Supervision
71		Limited CE access
72	24	Encryption
73	25	Effective human engineering
74		Clear, easy to use protection facilities
75		Adequate documentation
76		User education
77	26	⟨See features for threats 27-39⟩
78	27	Print log
79		Security conscious I/O routines

80	28	Print log
81	29	Print log
82		Security conscious I/O routines
83	30 31	Careful administrative procedures
84	32	Careful administrative procedures
85		Important mail sent registered or by courier
86		Delivery confirmation
87	33	Trace log of sensitive output
88		Library facility for sensitive output
89		⟨See also features for threats 34-37⟩
90	34	Clean desk policy
91		User education
92	35	Guarding work in transit
93	36	⟨Refer to features for threats 1-18⟩
94	37	Guarding work in progress
95	38	Buffer erase mechanism
96	39	Paper shredder
97		Use of old ribbons for sensitive jobs
98		Destruction of carbon paper and ribbons
99	40	⟨Refer to features for threats 41-44⟩
100	41	Careful administrative procedures
101		Important mail sent registered or by courier
102		Delivery confirmation
103	42 43	Second person verification
104		Checksums
105		Software checks
106	44	Verification checks
107		Checksums
108		Software checks
109		Originator verification
110	45	Testing
111		Audit programs
112		Testing and verification
113		Penetration attempts
114	46	Program testing and validation
115	47	Adequate documentation and design specs
116	48 49 50	Adequate documentation and design specs
117		Program testing and validation
118		Programmer education
119	51	Program testing and validation
120		Code inspection, recompilation
121		Choosing writer who could not benefit
122	52	Testing on setup data
123		Containment of test programs

A panel session—The contribution of planning to information systems productivity

SESSION CHAIRMAN—CHARLES C. TUCKER
Twentieth Century-Fox Film Corporation

Panel Members

John A. Zachman—International Business Machines
Daniel S. Appleton—Borg-Warner Corporation
Michael J. Kirrene—Avco Financial Services

PANEL OVERVIEW—Charles C. Tucker

The traditional approach to information systems development has been to plan at the project level. This leads to the development of independent systems, each with its own data, for each user. Most organizations suffer from the following shortfalls that results from this traditional approach:

1. Systems which are not supportive of business plans
2. Systems that are not easily adaptable to organizational changes
3. Data which is redundant and inconsistent
4. Systems that are difficult to interconnect
5. "Squeaky wheel" priority determination.

Management views this in terms of systems with short useful lives, which are expensive to operate and maintain, and support areas with little impact on the overall success of the company.

The Business Systems Planning methodology used at Fox overcomes these shortfalls by:

1. Tying information systems plans directly to business plans
2. Designing systems around business processes so they are organizationally independent
3. Treating data as a company resource so it can be shared and managed
4. Laying out the overall architecture or network first so the individual systems can be easily interfaced as they are developed
5. Providing a logical framework for determining implementation priorities.

Because of our investment in Business Systems Planning, we expect that the information systems we develop in the future will last longer, cost less to develop and operate, support business plans better, and address our most important information requirements first.

THE INFORMATION SYSTEMS MANAGEMENT SYSTEM: A FRAMEWORK FOR INFORMATION SYSTEMS PLANNING—John A. Zachman

Technology is precipitating a change in management's perception of the role of information systems (I/S) in the business environment. Business Systems Planning is an analytical methodology that has evolved in an attempt to better define this role and manage I/S more effectively in support of management.

As a result of a number of Business Systems Planning analyses, it is becoming apparent that the key to increasing I/S productivity in light of the new role being defined, is the recognition of several components of the overall I/S Management System that must be managed uniquely.

These components include:

1. A Strategic Plan and planning process that looks externally at the business, as well as integrating with the more typical Management Control and Operational Control plans and planning processes which look internally at I/S.
2. An Information Architecture that depicts the business and serves as the "preliminary design" of the integrated product that the I/S organization is in place to build.
3. A Data Architecture (and its attendant management facility) that protects the consistency of data for management's measurement purposes, as well as controls the project development activities to insure their conformance to the Information Architecture.

These components of the I/S Management System are over and above the traditional Development and Operations components, well recognized as requiring unique attention, supporting the role of I/S as it is perceived in today's typical environment. The recognition and management of the additional components will have a substantial impact in increasing productivity by minimizing systems redesign for data integration purposes, minimizing maintenance activity

caused by the interdependence of individual systems and files, and insuring that the systems developed more closely meet the expectations and requirements of management.

A DATA MODEL APPROACH TO BUSINESS SYSTEMS PLANNING AND CONTROL—Daniel S. Appleton

The management problem is to establish a viable Business Systems Plan based on the basic business mission. The MIS problem is to implement the Business System Plan. To do this, MIS must create a system structure in the business which will respond to constant changes. MIS must understand the fundamental structure and keep that structure flexible in response to the business.

At Byron Jackson Pump (BJP) we first developed a data model of our business structure.* Next, we tested this model conceptually to determine how it would respond to various changes: e.g., decentralization, product standardization, new product development, productivity improvement, cost reduction.

The data model of the business was then established as our basic Management Systems Technology and placed under control of what we call the Storage and Processing Control System (SPCS). The SPCS provides centralized, standardized control over all data elements, internal processing routines, minimum edits, security routines, and physical file structures. It defines the basic management technology which can be customized to the requirements of the various BJP manufacturing locations and DP Service Centers worldwide. It handles the basic spectrum of manufacturing systems, from process control through job shop control, including any hybrid mix of the two, and accommodates fluctuations based on product mix changes and strategic plans.

The level of management technology of the SPCS is controlled by the MIS Steering Committee, which funds major modifications. Minor modifications required to customize for efficiency or effectiveness at a given location are controlled by the location's System Management Team.

BJP's two other MIS control systems, i.e., the Input and Output Control Systems, provide the basic tools for distribution, implementation, and customization of the management technology of the SPCS at individual locations. The I/O Control Systems are managed by DP Service Centers in coordination with local Systems Management Teams in each location.

All three control systems taken together comprise BJP's Functional Network (FN) concept. FN is an approach to distributed data processing which concentrates on improving the efficiency and cost-effectiveness of the three basic DP functions: (1) Storage and Processing, (2) Input, and (3) Output. Its advantages over the traditional Applications Network concept include:

1. Lower DP marginal costs
2. Increased compatibility with business growth and dynamics
3. Better control over DP technology
4. More efficient use of DP overhead
5. More effective use of direct DP expenditures
6. Easier to customize
7. Reduced implementation time
8. Higher quality systems
9. Easier to maintain systems.

In summary, the data model describing BJP's management system technology provides a consistent, comprehensible foundation for development of the business system structure. Through BJP's Functional Network, the development, maintenance, implementation, customization and distribution of that management systems technology is controlled based on the specific needs of individual locations.

INFORMATION SYSTEMS PLANNING IN THE NON-PLANNING ENVIRONMENT—Michael J. Kirrene

Avco Financial Services (AFS) is the third largest consumer finance company in the U.S. with sizable operations in Canada and Australia and footholds in Puerto Rico, Great Britain, and Japan. The management team is young and aggressive, operations oriented, and determined to be number one. By any measures AFS is a successful company.

From a planning point of view, however, AFS represents the "typical" organization as defiend by John Zachman. Corporate planning and controlling efforts are focused upon individual projects. The formal Business Systems Plan as defined in the literature does not exist, and attempts to develop one would be viewed as bureaucratic endeavors designed to further delay addressing immediate data processing needs. While a company can obviously exist without a formal plan, the Information Systems Department must have a plan tied to the business goals in order to maximize its contribution to the welfare of the company it serves. It is even possible with todays operations support systems that data processings activities (or lack of activities) can be detrimental to the business.

The challenge is how to develop effective planning in a non-planning environment. Our Information Systems Department has taken a unilateral approach, including senior management whenever and wherever possible, but always taking the initiative. The results have been mixed. Although we do not have a Business Systems Plan, we at least have a projects plan reviewed by senior management on a quarterly basis and, more importantly, have taken that first crucial step of planning together with the company.

Intellectually, it is easy to accept the need for a Business Systems Plan properly derived from a careful analysis of the functional processes and information flow of the business. In the real world, however, it appears that rather than attempting to sell the typical company on Business Systems Planning, the Information Systems Department should be acting as a catalyst, developing business planning sponsors, aligning data processing more and more with the business objectives by learning more about the business it serves, and aggressively participating in the evolution from project planning to Business Systems Planning.

* See "An Approach to Manufacturing Automation," *Datamation*, Oct. 1977.

A panel session—Organizational factors in the allocation of computing resources

SESSION CHAIRMAN—ROB KLING
University of California, Irvine

Panel Members

William Dutton—University of California, Irvine
James Danziger—University of California, Irvine
Einar Steffereud—Network Management Associates
J. A. Sutton—IBM Corporation

A POLITICAL PERSPECTIVE ON COMPUTERS IN LOCAL GOVERNMENT—William Dutton

Most people concerned with computing in complex organizations do not consider computers a subject for political analysis; but few people can maintain this view in the future. Social scientists define "politics" as the allocation of goods, services, values, and resources. Thus every organization is faced with "political" decisions when deciding whether to fund one program rather than another. From this point of view, "politics" is neither inherently "dirty" nor restricted to institutionalized procedures, such as voting, but is a common element of organizational life. Computing is a political technology because it can, and frequently is, used to effect the distribution of values among various interests. In any organization there are serious differences of opinion and interest over the adoption, support, use, and control of computer technology.

The political elements of computing can be addressed in three areas: (1) the allocation of computing resources as a means to control other related resources (e.g., staff); (2) the computer's use to effect the distribution of information, and therefore the distribution of power and control in an organization, and (3) the computer's use to reinforce an organization's political economy as reflected in the kinds of goods and services it provides. This talk draws upon data collected by the URBIS Group in several national surveys and case studies of computing in American local governments since 1974.

We have found computing to be a complex resource "package." And the allocation of the resources comprising the computing package is often a significant issue within organizations. Directly, computers constitute a resource because they bring with them differential control of their capabilities vis-a-vis other resource controllers. Indirectly, computers raise political issues because they frequently affect the relative distribution of other resources such as staff, money and status amongst organizational units.

Computing also raises political issues within organizations because of its instrumental value to different actors, particularly as a result of information that can be provided. Top decision-makers are interested in information that might help clarify choices and identify problems. Managers and supervisors are interested in operational data about those whose activities they guide and direct. In public agencies, data-dependent departments like planning seek some of their data from information-generating departments, which sometimes place propietary rights on their data. Since computers frequently alter the content, direction, and speed of information flows, they thereby affect the relative influence of these organizational actors.

Many problems with computing are not a function of inadequate or unavailable technical or managerial solutions. Rather, they are due to differences of interest or opinion regarding potential impacts. Beliefs about the best solution vary accordingly. However, solutions to computing problems which are solely technically-based or management-based may fail because they ignore the local politics of computing. Furthermore, some problems lack solutions, *even when politics is considered.* Technical and managerial solutions may create politically unacceptable impacts. Or, politically attractive solutions might not align with feasible technical and managerial solutions.

These observations will be illustrated with a variety of data drawn from the URBIS studies.

SERVICE PROVIDER OR SKILL BUREAUCRACY?—THE DATA PROCESSING FUNCTION IN LOCAL GOVERNMENT—James Danziger

A straightforward interpretation of the data processing unit's function is that is a *provider of services* to its clients. Increasingly, the formal information systems of many organizations is centered in EDP. Data processing is responsible for providing the local government with a set of service activities involving data: record-keeping, record-searching, printing and calculating, record restructuring, and analysis. The quality of the service hinges on the effective utilization

of hardware, software, and staff skills to the accurate, useful, and timely treatment of data. In this analytic sense, the EDP function is similar to the typing pool (although computing is far more technical): both EDP and typing involve the skillful organization of data, to facilitate their use by other organizational staff. If one were to provide a normative theory of organizational roles relating to EDP as a service provider, top managers would have the primary responsibility for deciding what EDP should do, the EDP unit would provide these data handling services, and the users of EDP services would organize them for effective utilization.

Alternatively, the EDP unit can be characterized as a "skill bureaucracy." The distinguishing feature of a skill bureaucracy is its relative monopoly on some area of technical expertise. While it does provide services, its behavior is driven by certain self-oriented values: (1) to preserve autonomy from outside control; (2) to dominate its relationship with its clients; (3) to expand its domain of activity; and (4) to be guided by its internal standards of professionalism. There is good evidence, for example, that skill bureaucracies in education and public-welfare achieve these values relative to policy-makers and to clients of their own services. Given the highly technical nature of computing and the professional concerns of computing specialists, EDP units seem particularly likely to manifest attributes of a skill bureaucracy.

The concepts of service-provider and skill bureaucracy are ideal-type characterizations, and actual EDP units display some aspects of each kind of orientation. However, data drawn from surveys of computing practices in local government under the URBIS project indicates that skill-bureaucratic features are common in many American local government EDP organizations. This talk will address these alternative models of the EDP function based upon both survey data and case studies.

USEFUL APPLICATION OF POLITICS IN COMPUTING—Einar Steffereud

THE ORGANIZATIONAL ENVIRONMENT

In many large organizations computing allocations are made either by committee or by a "computing czar." While these choices seem to span the range of centralized authority through participatory decision-making, in fact they often fail to achieve the intended effects. This talk provides a framework for understanding how committees can be effectively organized to deal with different computing rationales used by operating units, and thus to act so as to sensibly distribute resources rather than simply diffuse responsibility.

Computing has assumed a central role as a major engine of production and is shared by large portions of many organizations. Despite the deployment of mimicomputers, many organizational units are becoming increasingly dependent upon shared or common computing systems (including networks). While it is common for different organizational units to develop their own rationales for action, shared data and computing resources tends to lead to smaller differences of rationale at the operating levels of large organizations.

PROBLEMS OF COMPUTER GOVERNANCE

As computing becomes more pervasive, it becomes more essential to find workable compromises between value system differences between operating departments in an organization. These negotiated compromises are essentially "political" rather than "technical" solutions. (Conflicting values cannot be forced by the authority of a computing "czar." The half-life of a computing "czar" is inversely proportional to the extent to which he assumes control of other people's resources.)

So how can the problems of computing governance by effectively resolved? An effective governance structure must aggregate the required power from operating departments. This differs from a responsibility diffusion committee which includes "representatives" from operating department who have little power to commit their departments to authoritative policies. Observations in several diverse computer using organizations indicate that user committees which fail to mirror the de facto power structure of the organization tend to be ineffective in dealing with conflicts over computing use, and fail to make "sensible" allocation decisions. With a power aggregation committee, substantial political issues can be resolved, which otherwise tend to be analyzed, debated, and avoided.

This talk illustrates several different strategies of computing governance with examples drawn from a variety of computer-using organizations.

ORGANIZATIONAL CONSIDERATIONS IN D.P. RESOURCE ALLOCATION—J. A. Sutton

Data processing resources frequently are in great demand by "user-departments" who would like to utilize those resources. When the allocation of these resources is perceived as unfair, members of user-departments may become dissatisfied with the data processing department. The frequency with which articles appear pointing out user-D.P. problems and suggesting solutions to these problems suggests dissatisfaction is widespread.

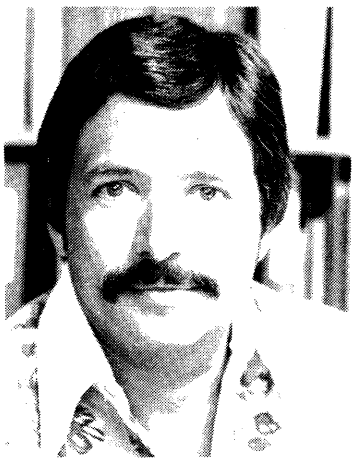
This discussion focusses on organizational mechanisms intended to provide the necessary communication and resource allocation between the Data Processing Department and departments wishing to utilize data processing services. Several mechanisms frequently proposed to facilitate interdepartmental transactions will be discussed, research results on their effectiveness will be reported, and recommendations will be made for the management of the interdepartmental interface.

Mechanisms for the management of the interface between

D.P. and user departments generally fall into two categories: (1) formalization of interfaces (e.g., establishment of a D.P. Steering Committee), and (2) decentralization of some function to the user department (e.g., location of systems analysts in the user department). Mechanisms from both of these categories are frequently intended to "increase user involvement."

Organizational behavior theory suggests that the potential for conflict at the D.P.-user department interface is great and that substantial benefits might accrue from improved integration between departments. Recent research suggests that some of the mechanisms proposed to enhance D.P.-user relations have little impact (e.g., whether the D.P. manager reports to a department head or is himeslf a department head has little impact on user satisfaction with D.P.). We observe that as D.P. departments grow in size, they exhibit a tendency to increase the number of formal mechanisms in place, yet evidence suggests that simply adding mechanisms has little effect on user satisfaction. However, the perceived effectiveness of many mechanisms is closely related to user satisfaction with D.P.

These results suggest that adding mechanisms intended to enhance the perception of "fair" resource allocation and to improve user satisfaction with D.P. may not serve the intended purpose. Rather, the results suggest that the *quality* (or perceived effectiveness) of interdepartmental integration mechanisms may be the critical issue. Hence, the efforts of the D.P. manager (and others) may be better spent at improving the effectiveness of existing mechanisms rather than putting new mechanisms in place.



Area Director:
Peter Freeman
SofTech, Inc. and
University of California at
Irvine (on leave)

Software development methodology

OVERVIEW

Beginning in the late 1960s with an interest in programming methodology, researchers and practitioners alike have begun to pay increased attention to the methods used to develop software. This interest in technique is driven by two major concerns: productivity and quality.

As the cost of software has soared, both in absolute terms and as a fraction of the budget of many organizations, managers have realized that the productivity of the software development group was a critical factor. The call has gone out for improvement and this in turn has caused many people to take a new look at the methods they are using.

The vastly expanded role of software-intensive systems in the operation of most organizations has also caused managers and users alike to look critically at the quality of the software produced. The most obvious quality factor is reliability, but there are many other factors such as security, efficiency, and maintainability. As with productivity, those responsible for building software have realized that one of the variables in building quality software is the method used.

This increased emphasis on methodology has rapidly expanded beyond the original concern with programming alone. Today, we realize that the entire life-cycle of software—from initial need through years of repair and enhancement—must be considered when we start to address issues of quality and productivity.

Matching this life-cycle awareness, we have organized five sessions for NCC '78 that address methodological questions in different phases of software development:

Software Design and Analysis includes papers that address questions of overall system definition and organization.

Programming Methodology will focus on several topics that are relevant to the way in which we create individual programs.

Software Verification, Validation, and Testing samples several new and important concepts in the still-critical area of demonstrating system correctness.

Software Maintenance has been organized especially to address this phase that is often neglected in technical meetings. Although no papers appear in the Proceedings, the panelists represent a large amount of experience with maintenance activity, making this an especially interesting session.

User Experience with New Software Methods is an attempt to redress an imbalance inherent in technical meetings. A panel of six experts with personal and corporate experience using many of the new methods has been assembled. They will share their experience as users—not creators—of the new methods and discuss some of the problems encountered and benefits gained in implementing them. This should be an especially valuable session to attend, in addition to reading the position papers from the panelists that appear in the Proceedings.

All of the paper sessions include a discussant to provide balance and perspective on the presentations. We hope that these sessions will contribute to your understanding of new software methodologies and, in turn, help you and your organization to improve both the quality and productivity of your software development.

A description scheme to aid the design of collections of concurrent processes*

by WILLIAM E. RIDDLE
University of Colorado
Boulder, Colorado

JOHN H. SAYLER and ALAN R. SEGAL
University of Michigan
Ann Arbor, Michigan

ALLAN M. STAVELY
New Mexico Institute of Mining and Technology
Socorro, New Mexico

and

JACK C. WILEDEN
University of Massachusetts
Amherst, Massachusetts

INTRODUCTION

Abstract data types have recently emerged as an important facility for the specification of sequential programs. Originally defined in the language SIMULA,[1] abstract data types were soon recognized as useful in structured programming.[2] Their recent refinement has taken three directions. First, they have been developed to support top-down design methods.[3] Second, they have been extended so as to provide a basis for formal verification.[4] Finally, they have been used as the basis for a tool to aid program development[5] which admits rigorous, but perhaps incomplete, analysis.[6] The variety of these uses indicates the breadth of the benefits which accrue from facilities for high-level, abstract description.

While abstract data types are convenient for the description of a software system's data storage components, they are not convenient for the succinct description of those system components which are more for the processing than the storage of data. This is particularly true when the components operate concurrently.** The major problem is that abstract data types are oriented towards describing components as structures of data which are operated upon via procedure calls. Many components—e.g., a text editor in an operating system or a file system in a multiprocessor computing facility—are not naturally described in this manner.

Extensions and modifications to abstract data types could be developed to increase their effectiveness in describing processing components in software systems—this has been done in the Gypsy system[7] and the Modula programming language.[8] In this paper, however, we develop a description scheme that retains many of the concepts of abstract data types but builds upon the concept of a sequential process. Further, the description scheme developed in this paper focuses upon the succinct *modelling* of a system's processing components rather than their detailed programming.

The modelling scheme presented here is an extension of a scheme for describing sequential process interaction[9] that was developed as a basis for software system analysis.[10] It has been prepared for use in the Design Realization, Evaluation And Modelling (DREAM) system[11] which provides bookkeeping and analysis aid to the designers of large-scale software systems. The scheme is part of the DREAM Design Notation (DDN) which has been developed to permit the incremental development of a system's design as a series of design description fragments, called *textual units*.

In the next section, the basis for the modelling scheme is established by developing an abstract view of sequential process interaction as occurring through message interchange. The following sections then develop the model of

* This work was supported by a grant from Sycor, Inc.

** By *concurrent* we mean parallelism which may be actually achieved by executing the system in a multiprocessing environment or which may be only apparent at abstract levels of system description and never achieved during system execution.

a *subsystem,* i.e., a collection of concurrent sequential processes. Subsystem interfaces and interaction are described first. Then a scheme for the non-procedural description of a subsystem's behavior is presented. The paper concludes with a brief discussion of the ways in which the modelling scheme may be beneficially used during the development of large-scale software systems.

The focus of this paper is upon DDN constructs for describing collections of concurrent sequential processes. More complete treatment of this material and discussions of other aspects of DREAM may be found in References 12 through 16. Also, the focus is upon the use of the constructs—their syntax is covered in Reference 17. Some justification for the constructs are given; others lie in the DREAM system's general philosophy which is discussed in Reference 18.

AN ABSTRACT VIEW OF SOFTWARE SYSTEMS

A software system may be viewed as composed of parts, *subsystems,* which operate concurrently and asynchronously and which interact either through shared data objects or by the sending and receiving of messages. This is a natural way in which to view multiprocessor systems, since these systems actually have components which interact by message transmission. But this view is also appropriate when it is merely a logical one and the system actually runs in a uniprocessor environment. In this case, the view facilitates the decomposition of the system (and thereby the mastery of its complexity[19]) and message transmission is used only to model the actual interactions.

As an example of this view of software systems, consider the HEARSAY speech recognition system[20] developed at Carnegie-Mellon University. One subsystem within the HEARSAY system is a data base, called the blackboard, which contains all the information about the utterance being recognized and the hypotheses which have been made as to its linguistic structure. The other subsystems within the HEARSAY system are processing components called knowledge sources. Each knowledge source inspects the information in the data base and augments or modifies it according to the rules which the knowledge source is programmed to enforce.

The interactions among these subsystems are as follows. Since it would be wasteful to have each knowledge source constantly inspect the data base to determine whether it should perform any processing, the data base is programmed to know which data base entries are of interest to each knowledge source and to send a signal to a knowledge source whenever one of the data base entries of interest to it changes value. Each knowledge source is reentrant and this (conceptually) means that each knowledge source has several internal servicer subsystems, each of which can perform the processing to be done by the knowledge source. When a knowledge source is signalled by the data base, one of these servicer subsystems is activated and it inspects the data base and makes any appropriate modifications. The interactions between a servicer and the data base consist of a sequence of message transmissions, with the data base returning messages in response to the requests sent by the servicer.

THE MODELLING SCHEME

A modelling scheme based upon this abstract view of software systems must have facilities for describing subsystems and their interactions. In this paper, we focus upon those subsystem interactions which occur through message transfer. We assume, therefore, the existence of a scheme for describing data objects that may be shared by a community of concurrent subsystems.* Such a scheme must be able to describe many aspects of shared data objects, but for the present discussion it is only important that the scheme allow the specification of the possible values of the data objects.

A subsystem model must describe three aspects of the subsystem. First, it must specify the interfaces to the subsystem not only in terms of the format of the information flowing through each interface but also in terms of what information may legally flow through each interface. Second, it must specify the legal sequences of message flow through the interfaces. Finally, it must relate the subsystem's operation at one point in time to its operation at a previous point in time—i.e., it must specify the more global aspects of the subsystem's operation.

In the following sections, each of these aspects is discussed and illustrated by examples which, taken together, comprise a description of the knowledge source subsystems of HEARSAY.** Since all of the knowledge source subsystems are similar with respect to their interactions with the data base, the description developed in the examples is of the *class* of knowledge source subsystems. To reflect that different instances of this class vary with respect to their details, such as the number of entries in the data base that are of interest to the knowledge source, we define a parameterized class. This means that part of the definition of the class is a specification of *qualifiers* which may be assigned values when an instance of the class is created. Class definitions and qualifiers are discussed more fully in Reference 12.

SUBSYSTEM INTERFACES

A subsystem's interface consists of a set of *ports* through which messages may flow. Conceptually, a port is a communication line along which messages flow, one at a time, and which does not have any storage capabilities. Ports correspond to uni-directional communication lines and therefore have a direction *in* or *out.*

---

* A scheme that was developed in conjunction with the work reported here is discussed in Reference 13.

** The description is an approximation of the actual structure and operation of the knowledge sources in HEARSAY. It reflects our understanding of the description which appears in Reference 20, but has also been constructed so as to provide examples of the facilities in DDN.

```
[knowledge_source] : SUBSYSTEM CLASS;

  QUALIFIERS;
     DOCUMENTATION;
        #_values is the number of data base entries
        monitored for this knowledge source; #_servers
        is the number of parallel servers in this
        knowledge source
        END DOCUMENTATION;
     #_values, #_servers
     END QUALIFIERS;

  await: ARRAY [1::#_values] OF IN PORT;
     BUFFER SUBCOMPONENTS;
        signal OF [on_off_switch]
        END BUFFER SUBCOMPONENTS;
     END IN PORT;

  make_request: ARRAY [1::#_servers] OF OUT PORT;
     BUFFER SUBCOMPONENTS;
        request OF [data_base_operation]
        END BUFFER SUBCOMPONENTS;
     BUFFER CONDITIONS;
        request=inspect,
        request=modify
        END BUFFER CONDITIONS;
     END OUT PORT;

  get_answer: ARRAY [1::#_servers] OF IN PORT;
     BUFFER SUBCOMPONENTS;
        answer OF [data_base_response]
        END BUFFER SUBCOMPONENTS;
     END IN PORT;

  END SUBSYSTEM CLASS;
```

Figure 1

The messages which flow through a port are (ordered) sets of data objects. Each port therefore has associated with it a set of *buffers,* each able to store one data object. The set of buffers indicates the types of data objects which comprise a message and the order in which the data objects are composed to form a message.

The messages which may legally flow through a port are specified by giving *buffer conditions* for the port. A buffer condition is a predicate over the buffer data objects, indicating the set of legal values for the buffer data objects as well as the legal correlations among the values. OR'ed together, the buffer conditions associated with an in-port (out-port) are analogous to a pre-condition (post-condition).[21]

In DDN, a port is defined by a textual unit which specifies the port's name and direction and which has nested textual units which specify the buffers and the buffer conditions associated with the port. A set of ports is defined in Figure 1 for objects in the class [knowledge_source].* If an object of this class were created with #_values having the value 4 and #_servers having the value 3, then the object would appear as pictured in Figure 2. Notice that defining an array of ports implies that there is an array of buffers, one for each port. The condition associated with the make_request ports indicates that messages flowing out through these ports should have only the value *inspect* or *modify*.

* It is a convention in DDN to enclose an identifier in square brackets when it is used to name a class.

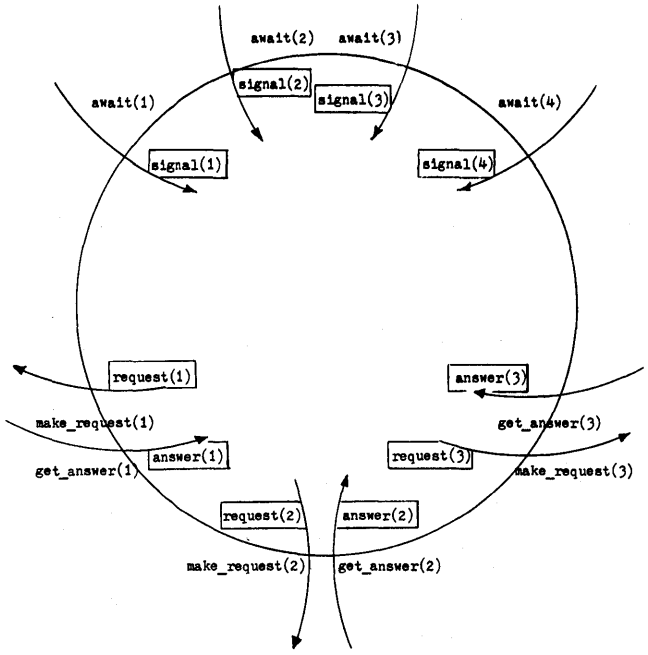


Figure 2

A port definition is comparable to a heading for a procedure stated in a programming language. It is similar in that it specifies a name by which the port (procedure) may be referenced and the number, type and order of the data objects (parameters) in messages (parameter lists) processed by the internal components (procedure body). It differs because ports allow only one-way communication whereas procedures provide two-way communication. The buffer conditions of DDN are similar to the *entry* and *exit* specifications of Alphard[4] with the additional aspect that they are required to be valid whenever a message flows through the port.

MESSAGE FLOW THROUGH A SUBSYSTEM

The role that a subsystem plays within a community of subsystems is specified by a definition of the correlations among the messages flowing into and out of the subsystem. When this defines the injection of messages into the subsystem's environment as a result of the reception of messages, then it serves to specify the facilities provided by the subsystem. When it defines the reception of messages subsequent to the injection of messages into the environment, then it serves to specify the subsystem's utilization of other subsystems in its environment.

A subsystem's message flow characteristics may primarily be defined in terms of sequences of message transmissions through the subsystem's ports. This is analogous to defining a procedure in terms of a set of parameter/result pairs. More complex characteristics, called *global characteristics,* concern the correlations between transmissions occurring in different message transmission sequences. For example, in manipulating a stack, a pop operation must be preceded, at

some point in time, by a push operation. In this section, the concern is with the more easily specified sequential message transmission characteristics. The specification of global message flow characteristics is treated in the next section.

Sequential message transmission characteristics are specified by giving a set of programs, each of which is a model of a sequential process, called a *control process*. Each control process model specifies a set of sequences of message flow through the subsystem. A control process is defined by a nondeterministic, procedural model which specifies the control process' message reception and transmission activity. Nondeterminism is allowed because it contributes to the clarity of the model, allowing succinct definition of the control process. A procedural specification is also used to enhance the clarity of the description.

Messages flow in and out through a subsystem's ports as a result of *receive* and *send* operations. For a send operation, a message is first composed using the values of the buffers associated with the port that is specified in the send operation. Then, the message is sent out through the port to be placed in the link to which the port is attached* and thereby made available for reception by some subsystem. The control process which invoked the operation is suspended while the message is constructed and placed in the link. Since links have infinite storage capacity, this suspension is relatively short and the send operation can therefore be considered to be a non-blocking operation.

When a receive operation is performed, the control process invoking the operation is suspended until a message is retrieved from the link to which the port specified in the receive operation is attached. The message is then decomposed and distributed among the buffers associated with the port. Since it is possible to request a message when none is available, the receive operation can block the control process which invoked it for relatively long periods.

Prior to a send operation, the subsystem will compute the values of the data objects which compose the message and place these values in the buffers. This computation may be lengthy and complicated, but neither its time consumption nor its detail are of interest in defining sequential message

```
'[knowledge_source]: SUBSYSTEM CLASS'
  listener: ARRAY [1::#_values] OF CONTROL PROCESS;
     MODEL;  ITERATE
                RECEIVE await(MY_INDEX);
                END ITERATE;
             END MODEL;
     END CONTROL PROCESS;
```

Figure 3

* Communication among asynchronous message transmitters and receivers requires a transmission controller that is able to store messages that have been sent but not yet received and requests for messages which have been lodged but not yet satisfied. In DDN, an idealized controller, called a *link*, is provided. Links hold messages and requests in (unbounded) bag data structures. Thus they do not necessarily pass messages on in the order the messages were sent, nor do they necessarily service requests for messages in the order the requests were lodged. Ports may be attached to links by a process described in Reference 14.

```
'[knowledge_source]: SUBSYSTEM CLASS'
  servicer: ARRAY [1::#_servers] OF CONTROL PROCESS;
     MODEL;  ITERATE
                request(MY_INDEX) SET TO modify OR inspect;
                SEND make_request(MY_INDEX);
                RECEIVE get_answer(MY_INDEX);
                END ITERATE;
             END MODEL;
     END CONTROL PROCESS;
```

Figure 4

transmission characteristics. Thus, in a control process, computational detail is suppressed by modelling it with a *set-to* operation which may be applied to a buffer data object and causes the buffer to assume a value prescribed in the set-to operation.

To describe the sequential message transmission characteristics of [knowledge_source] subsystems, two control process types are needed. The first, specified in Figure 3,* models the consumption of messages which arrive at the await port from the data base. This models the subsystem's operation with respect to signals sent to indicate a change of some data base entry of interest to the knowledge source. The variable MY_INDEX has a value in the range declared as the bounds of the array of control processes and is used to make each control process model distinct with respect to the buffers and ports to which it refers.

The second type of control process is specified in Figure 4. These control processes model the operation of the servicers of the incoming signals, indicating that they present requests to the data base through one of the make_request ports and wait for answers to the requests through one of the get_answer ports. Note that nondeterminism may be specified in the set-to operation by giving a logical expression which specifies the set of values which could possibly be assigned to the data object as a result of the computation modelled by the set-to operation. The control process structure of a [knowledge_source] subsystem in which #_servers is 3 and #_values is 4 is graphically represented in Figure 5.

Each model is a control program over send and receive operations on ports and set-to operations upon buffers. The control constructs are Algol-like. There are constructs for definite iteration: an "ITERATE n TIMES" construct and a "FOR ALL i IN set_of_values" construct. WHILE and UNTIL constructs are available for indefinite iteration. Since many subsystems are designed to never terminate, there is also an ITERATE construct for infinite iteration. Conditional control may be specified by the usual forms of the IF construct. There is also a generalized CASE construct which allows "labelling" of the cases with logical expressions. All of the control constructs have a nondeterministic version. For example, the construct "ITERATE n OR MORE TIMES" may be used to indicate that the number

* The quoted prefix attached to the control process textual unit in Figure 3 indicates that this textual unit is intended to be an additional part of the definition of the [knowledge_source] subsystem class. This prefixing construction allows selective modification of a DREAM design description, thereby permitting incremental elaboration of a software system's design.

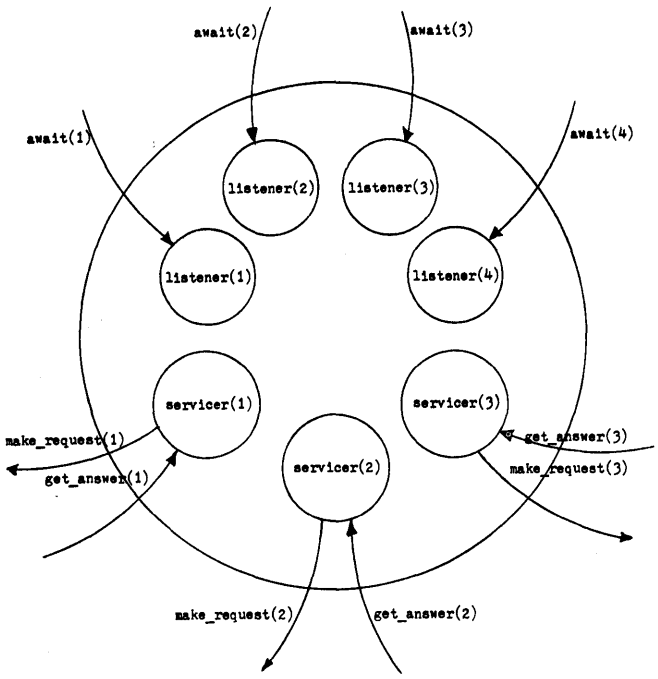


Figure 5

of times, while known before iteration begins, can be any number greater than or equal to n.

GLOBAL BEHAVIOR

The facilities provided by a subsystem generally cannot be used in a totally arbitrary order. For example, the facility which a file system provides for opening a file must be used prior to the facilities for reading and writing the file. Thus there are correlations between what happens in one sequential message transmission and what may happen in another sequential message transmission.

This global behavior may be specified in terms of *events*, activities that may be observed external to the subsystem. Usually an event is the transmission of a message through a port, and thus may be identified as the execution of a send or receive operation in one of the control process models. In general, an event may be associated with the execution of any one of the instructions in a control process model. (Since control process models are used as part of the behavior description of a subsystem, they are known to an external observer.)

```
'[knowledge_source]: SUBSYSTEM CLASS'
   listener: ARRAY [1::#_values] OF CONTROL PROCESS;
      MODEL; ITERATE
         hear:   RECEIVE await(MY_INDEX);
                 END ITERATE;
             END MODEL;
      END CONTROL PROCESS;
'[knowledge_source]: SUBSYSTEM CLASS'
   servicer: ARRAY [1::#_servers] OF CONTROL PROCESS;
      MODEL; ITERATE
         start:  NULL;
                 ITERATE WHILE PERHAPS
                    request(MY_INDEX) SET TO modify OR inspect;
         ask:       SEND make_request(MY_INDEX);
         get:       RECEIVE get_answer(MY_INDEX);
                    END ITERATE;
                 END ITERATE;
             END MODEL;
      END CONTROL PROCESS;
```

Figure 6

```
'[knowledge_source]: SUBSYSTEM CLASS'
   EVENT DEFINITIONS;
      consult: SEQUENCE(ask, get),
      hear_and_do_something:
            SEQUENCE(hear, start, REPEAT(consult))
      END EVENT DEFINITIONS;
```

Figure 7

Global behavior is specified by defining a set of sequences of events. Note that this is what was accomplished procedurally via the control process models—each model defines a set of sequences of message transmissions where each sequence corresponds to an execution of the model. For the non-procedural specification of behavior, formal language theory provides several techniques since the set of events may be considered to be an alphabet and a behavior is then a language over this alphabet.[10]

Two aspects of global behavior—one being required for correct operation and the other corresponding to a design decision—need to be specified for [knowledge_source] subsystems. First, a servicer must not interact with the data base until after it has been activated by a signal indicating that an entry in the data base has changed. Second, the interactions with the data base must be ordered such that no request from any of the servicers is lodged until the previous request has been answered—the servicers must coordinate their interactions with the data base so that they utilize its facilities in a subroutine fashion.

To describe this behavior, we first define some events as indicated in Figure 6. The statement labels define names for events which correspond to the execution of the labelled statement. Note the use of the NULL instruction to allow denotation of the event "a servicer starts a sequence of interactions with the data base."

More macroscopic events are needed to conveniently define the global behavior. These are defined by the textual unit shown in Figure 7. The REPEAT operator denotes that the consult event may be repeated zero or more times. The SEQUENCE operator denotes that the events which are its arguments are sequenced in the specified order. Thus, the hear_and_do_something event corresponds to a signal ar-

```
'[knowledge_source]: SUBSYSTEM CLASS'
   DESIRED BEHAVIOR;
      POSSIBLY #_servers CONCURRENT
         (hear_and_do_something, [knowledge_source]|
                                 hear_and_do_something),
      MUTUALLY EXCLUSIVE
         (consult, [knowledge_source]|consult)
      END DESIRED BEHAVIOR;
```

Figure 8

riving from the data base and the subsequent interactions of a servicer with the data base.

Using these events, the global behavior may be specified as in Figure 8. The first part of the specification indicates that hear_and_do_something events may proceed in parallel up to the limit imposed by having only #_servers servicers. The second part indicates that interactions with the data base are mutually exclusive, i.e., ordered in time.

This example has used only a portion of the facilities available in DDN for behavior specification. Events may be associated with other aspects of a subsystem's operation, more complex relationships among events may be defined, relationships between events defined for different subsystems may be established, and events which are not associated with any computational part of the system may be defined. A complete description of DDN facilities for behavior specification is given in Reference 15.

CONCLUSION

The modelling scheme presented in this paper allows the abstract description of a collection of concurrent processes (a subsystem). Models in this scheme describe the interactions which may take place between a subsystem and other subsystems. A model therefore provides a specification for a subsystem which describes its behavior in relation to its environment but hides the subsystem's operational detail. A model consists of a definition of the subsystem's interface, a procedural description of legal usage of the interface and a non-procedural description of the legal usage of the subsystem over time.

Models in the scheme are rigorous, unambiguous specifications of the components within a software system. At the very least, the models serve to record the facilities which each component is to exhibit and the manner in which these facilities are to relate. The models provide, therefore, specifications which may be used to guide the eventual implementation of the components.

The models may also be used in the formulation of arguments as to the appropriateness of a system's design. While general techniques for formal verification cannot be defined, both simulation and analytic techniques can be used to derive information concerning the dynamic characteristics of the interactions among the subsystems.[22] The designer may then use this information to determine whether or not the specified desired behavior is actually achieved or to uncover situations in which the desired behavior is not achieved. This allows the designer to make corrections before proceeding and to proceed with increased confidence in the validity of the design.

We have found the modelling scheme to be convenient for the description of a variety of software systems.[23-27] We feel that it demarcates an important set of facilities which are required for the specification of software systems and that it will prove to be valuable as a basis for a variety of tools to aid designers of large-scale software systems.

REFERENCES

1. Dahl, O. and K. Nygaard, "SIMULA—An Algol-Based Simulation Language," *Comm. ACM, 9,* September 1966, pp. 671-678.
2. Hoare, C. A. R., "Notes on Data Structuring," in Dahl, Dijkstra and Hoare, *Structured Programming,* Academic Press, New York, 1973.
3. Liskov, B. H. and S. N. Zilles, "Specification Techniques for Data Abstractions," *IEEE Trans. on Software Engineering, SE1,* 1, March 1975, pp. 7-19.
4. Wulf, W. A., R. A. London and M. Shaw, "Abstraction and Verification in ALPHARD: Introduction to Language and Methodology," Report 76-46, Information Sciences Inst., Univ. of Southern Calif., 1976.
5. Henderson, P., et al., "The TOPD System," Tech. Report 77, Computing Laboratory, Univ. of Newcastle upon Tyne, England, September 1975.
6. Henderson, P., "Finite State Modelling in Program Development," *Proc. 1975 International Conf. on Reliable Software,* Los Angeles, April 1975.
7. Ambler, A. L., et al., GYPSY: A Language for Specification and Implementation of Verifiable Programs," ICSCA-CMP-2, Certifiable Minicomputer Project, Univ. of Texas, Austin, January 1977.
8. Wirth, N., "Modula: A Language for Modular Multiprogramming," *Software—Practice and Experience,* 7, 1977, pp. 3-35.
9. Riddle, W. E., "The Hierarchical Modelling of Operating System Structure and Behavior," *Proc. ACM 72 National Conf.,* August 1972.
10. Riddle, W. E., "An Approach to Software System Modelling, Behavior Specification and Analysis," RSSM/25, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, July 1976.
11. Riddle, W. E., J. Sayler, A. Segal and J. Wileden, "An Introduction to the DREAM Software Design System," *Software Engineering Notes,* 2, 4, July 1977, pp. 11-23.
12. Riddle, W. E., "Hierarchical Description of Software System Structure," RSSM/40, Department of Computer Science, Univ. of Colorado, Boulder, October 1977.
13. Riddle, W. E., "Abstract Monitor Types," RSSM/41, Dept. of Computer Science, Univ. of Colorado, Boulder, in preparation.
14. Riddle, W. E., "Abstract Process Types," RSSM/42, Dept. of Computer Science, Univ. of Colorado, Boulder, November 1977.
15. Wileden, J., "Behavior Specification in a Software Design System," RSSM/43, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, in preparation.
16. Segal, A., "Design Description Management," RSSM/45, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, in preparation.
17. Riddle, W. E., "DDN User's Guide," RSSM/37, Dept. of Computer Science, University of Colorado, Boulder, in preparation.
18. Sayler, J., "Philosophy of the DREAM System," RSSM/39, Dept. of Computer and Communication Sciences, Univ. of Michigan, in preparation.
19. Simon, H. A., "The Architecture of Complexity," *Proc. Am. Phil. Soc., 106,* December 1962, pp. 467-482. Also in Simon, *Sciences of the Artificial,* MIT Press, Cambridge, Mass., 1969.
20. Fennel, R., and V. R. Lesser, "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II," *IEEE Trans. on Computers, C-26,* 2, February 1977.
21. Hoare, C. A. R., "An Axiomatic Basis for Computer Programming," *Comm. ACM, 12,* 10, October 1969, pp. 576-580, 583.
22. Riddle, W. E., "A Formalism for the Comparison of Software Analysis Techniques," RSSM/29, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, July 1977.
23. Cuny, J., "A DREAM Model of the RC 4000 Multiprogramming System," RSSM/48, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, July 1977.
24. Stavely, A. M., "DREAM Design Notation Example: An Aircraft Engine Monitoring System," RSSM/49, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, July 1977.
25. Wileden, J., "DREAM Design Notation Example: Scheduler for a Multiprocessor System," RSSM/51, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, October 1977.
26. Segal, A. R., "DREAM Design Notation Example: A Multiprocessor Supervisor," RSSM/53, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, August 1977.
27. Cuny, J., "The GM Terminal System," RSSM/63, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, August 1977.

On the construction of interactive systems*

by MARTIN FREEMAN**
Stanford University
Stanford, California

and

WALTER W. JACOBS
The American University
Washington, DC

and

LEON S. LEVY
University of Delaware
Newark, Delaware

INTRODUCTION

Computer-based systems that perform sophisticated tasks in the face of challenging and variable environments are being developed in increasing numbers. Among the examples that can be cited are space exploration modules, highly automated weapon systems, and data-base management systems. The design and development of systems of this class involve many difficulties, and no generally accepted, successful methodology for the task is available.

This paper proposes a contribution to such a methodology in the form of a model that is believed to be generally applicable to the construction of interactive systems. The model is based on the identification of key features common to such systems. These features will be described first. To illustrate the concepts involved, running examples of a large timesharing operating system, and a robot driving an automobile in traffic will be used. Later, applications of the model to the design of microprocessor systems and multilanguage processing systems will be discussed.

INTERACTIVE SYSTEMS

Interactive systems are complexes of people, hardware, and software, performing a set of well-defined *tasks*. These tasks are subject to, or initiated by, conditions met in the *environment,* and are carried out under general *performance objectives*. A timesharing operating system has an environment consisting of user-manned terminals and other peripheral devices, and its tasks are the standard types of user requests. A robot chauffeur has an environment consisting of its passengers and the world outside the car, including roads, other cars, traffic signals, parking areas, and so on. Some of its tasks are starting, entering and leaving traffic lanes, parking, and obtaining fuel. Performance objectives for the robot include safety, comfort of passengers, and obedience to traffic regulations while for the timesharing system they include response time, throughput, and reliability.

As these examples show, tasks are the standard components of system activity. Tasks are refined by breaking them down into common *subtasks,* which are further decomposed until they result in relatively low-level units called *actions*. For example, the robot chauffeur's activity is expressed in terms of actions such as turning the ignition on or off, accelerating, steering and braking. These actions are the elements of the subtasks of turning a corner, obeying a traffic signal, occupying a parking place, and so on. Actions in the case of the timesharing operating system include compiling a program, searching a directory, or moving a block of data to permanent storage. Actions require *resources* for their execution. For example, an action for the operating system may be programmed as a machine language subroutine, but a processor and some core memory must be assigned in order that the action can be performed.

An interactive system may also receive assignments of unfamiliar tasks which must be carried out by *plans*—combinations of standard tasks which are formulated during the operation of the system. A plan will require a decision as each task is completed about the next task to be undertaken. A plan is needed by the robot chauffeur for an auto trip to an unfamiliar destination, while the operating system needs a plan for a large job, involving a number of tasks which must be completed quickly with minimum disruption to other system users.

An interactive system has *input* from, and *output* to, its

* The research reported here was supported by the National Science Foundation under grant MCS 76-07682.
** On leave from the American University.

environment. Input may include tasks, as well as data about the conditions in the environment that are involved in executing tasks. For example, users of the timesharing system can input requests, while the robot chauffeur may sense traffic conditions. Output may modify the environment or provide information related to tasks. The operating system may output an error message; the robot chauffeur may turn on the automobile's headlights.

Finally, an interactive system must store and maintain access to a considerable amount of *information*. Some of this information is general, concerned with the set of tasks and their translation into actions the system can execute. Another part of this information is environmental data—e.g., automobile traffic conditions for the chauffeur and job mix for the operating system. Yet another part is information which must be maintained about the states of the various parts of the system as well as planning information. The former might refer to the status of a task for a particular user of an operating system, while the latter might refer to information to be used to determine how well the chauffeur is performing an unfamiliar task assigned to it.

METHODOLOGY

In constructing a complex interactive system, one must find adequate solutions for three major aspects of system organization. First, the execution of system tasks must be controlled. The actual course taken in completing a task depends on the particular conditions met in the environment; and it can be recognized that the control problem is a substantial one. Second, the information needed by the system must be managed so as to be available when needed. And third, system resources (such as memory and processor time for an operating system) must be managed.

A complex interactive system must be organized into logical parts, and this must be done so as to meet two conflicting constraints. For economy, costly resources must be shared by system elements wherever possible, and functions that are the same should not be duplicated. But for effective performance, the parts should be as independent as possible, so as to avoid unnecessary communication and to keep down competition for resources, both of which add to the cost of overhead. A good trade-off between independence and sharing must be found if the design is to achieve performance objectives at reasonable cost.

One approach that has been used in constructing complex systems may be called *bottom-up*. Development begins with small elements that handle critical details of the system's operation. Larger aggregates are formed using these parts, and the process is continued. The resulting system is structured in layers, each with a particular capability; the top layer handles the tasks in the form that the system receives them. In the case of an operating system; this type of layered structure has been called a *hierarchy of virtual machines*.[1] This type of hierarchy can also be realized by an approach called *top-down* which starts with a gross description of the system and calls for successive refinement until the small critical elements of the system are reached.[2] The difficulty with the former approach is that considerations of performance and economy do not enter in the early stages of the development, and the early decisions may well make it impossible later on to achieve performance objectives economically. While this difficulty is removed in the latter approach, it is hard to apply unless one has a good understanding of how the system operates.

Another approach to construction is to dissect the system into *modules*, which operate independently except for well-defined controls and intercommunication; see e.g., Parnas.[3] The problem of the designer using this approach is to specify modules that require little duplication of information and low-level functions, and that can share resources to an appropriate degree. Useful guides to the choice of such modules do not seem to be available.

The proposed approach to modularization is intended to provide a guide or model of general applicability. Called *structural decomposition*, it derives from work on artificial intelligence.[4–7] It leads to breaking up the system into modules that either directly execute some action, or else support its execution. Support may take the form of control, allocation of resources, or management of information.

An important feature of the model is its separation of the short-range, intermediate-range, and long-range aspects of control. Short-range control is involved in executing and supporting individual actions. Intermediate control relates to the determination of the next action to be called when the outcome of the current action occurs. Long-range control arises in planning how to carry out assignments of unfamiliar tasks in terms of known tasks, in adapting the system to changes in the environment, or in otherwise modifying the system. Separation of control in this way reduces the average frequency of communication without restricting the kinds of communication that can take place, and can therefore lead to lower overhead.

Handling input and output in interactive systems often presents technical difficulties. Thus, the model places communication with the environment in a separate module.

STRUCTURAL DECOMPOSITION

An interactive system is constructed out of numerous components, with well-defined interfaces. Some parts are hardware, others are software; they are likely to be human components also. Hardware is usually specific to the particular type of system, but the other aspects of the modules that make up the system are being discussed. For the purposes of this exposition, it is irrelevant whether they are realized in software, firmware, or people.

The proposed approach to the design of such systems is to break them down into elements in a particular way. *Structural decomposition* implies that each module corresponds directly to some aspect of executing the task as a series of actions. The decomposition proceeds in stages, each stage refining the parts identified in the preceding one, until the system has been expressed as off-the-shelf elements.

The first two stages of this decomposition are general enough to serve as a model for a wide class of interactive systems. The description of these stages will emphasize the role played by each part in carrying out tasks, and also the communications between parts.

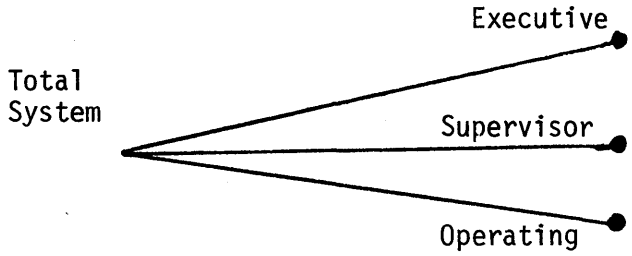


Figure 1—The first stage decomposition

At each stage, a specification for each part, the communications it receives and the communications it issues is determined. This specification guides the further refinement of the part, considered as a finite-state machine.

The first stage

The initial decomposition of the system is into parts that deal respectively with the long-range, mid-range and short-range aspects of activity. These parts are known as the *Executive, Supervisory,* and *Operating* components (see Figure 1).

The Executive is responsible for changes to the system, including both structural modifications and adjustments to adapt to variation in the environment. (This component usually is realized at least partly by a human administrator, but since changes to the system as initially put together are inevitable, the design should make explicit provision for them.) The Executive plans the expression of new tasks, if any, in terms of simpler tasks already known to the system. It also handles system start-up.

The Supervisor's job is to translate tasks into sequences of actions. It does this by initiating the next action on the basis of the outcome of the action just completed. The Supervisor also maintains knowledge of the current status of the system's tasks.

The Operator handles input from and output to the environment. Also, it carries out the actions called for by the Supervisor, making available resources and supporting functions as required. It keeps information pertaining to the environment and to the status of resources.

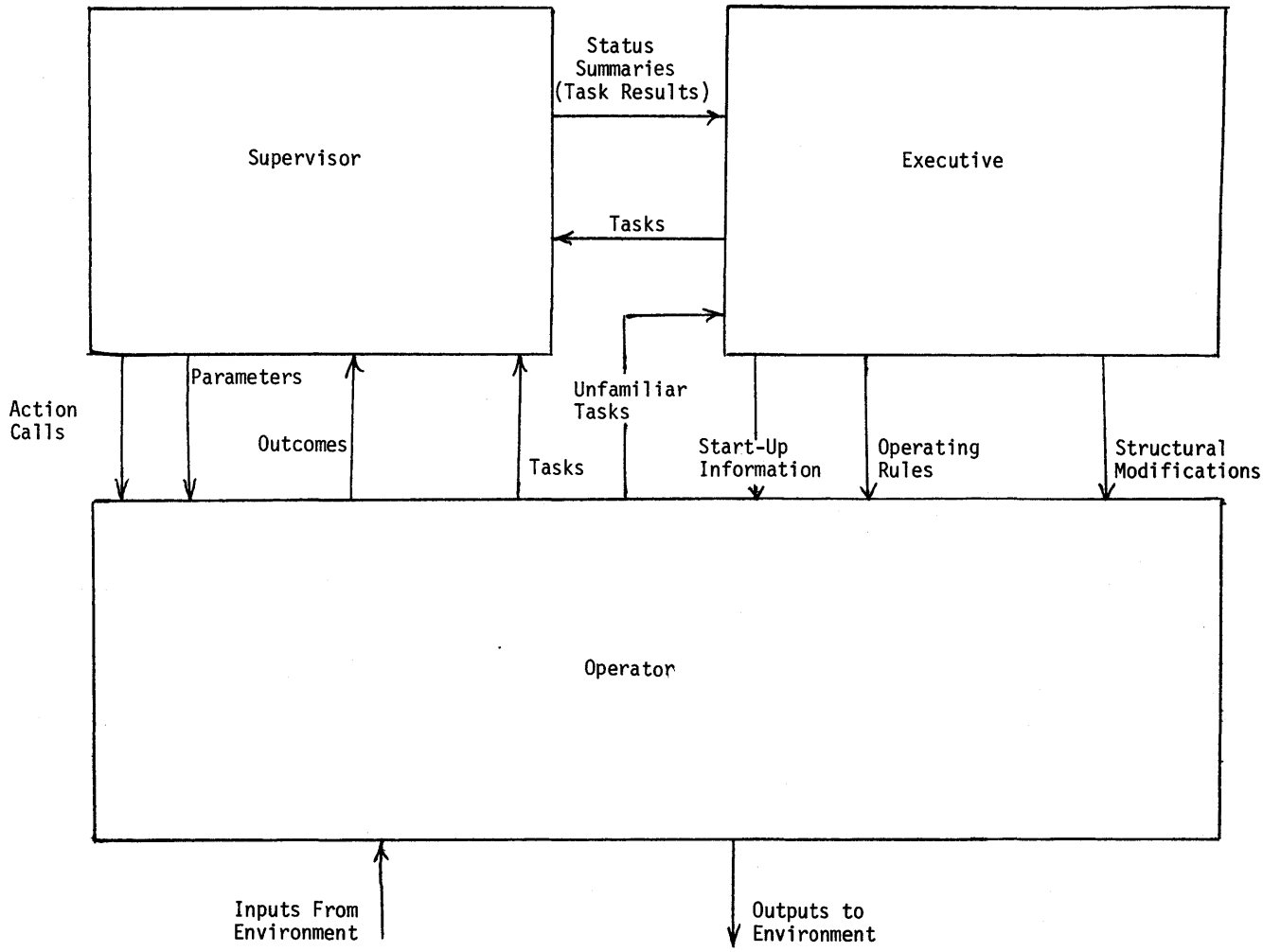


Figure 2—Communication in the model

Communications between these components must take place (see Figure 2). The Operator transmits to the Supervisor any tasks that it recognizes among the inputs from the environment. It also reports the outcome of each completed action, including any status information needed for specifying subsequent actions.

The Supervisor calls on the Operator for each action as appropriate, and provides the parameters that particularize the action to the current situation. Also, it provides to the Executive status summaries needed for the decisions the Executive makes.

The Executive supplies the Operator with start-up information, adjustments in operating rules, and structural modifications, as these become appropriate. If necessary, it passes to the Supervisor the task knowledge for handling a new task.

The second stage

Each of the components of the first stage is itself decomposed into parts. Among those parts are certain active elements that perform special functions. Such elements are known as *processors,* and indeed they may be realized by microprocessors; on the other hand, their functions may be supplied by programs that share general processing units. At the level of this discussion, it does not matter which way the element will be implemented. Also, it is possible that a processor is busy when a new call for its function arises. In such a case, the processor is assumed to have its own queue.

The "knowledge" or "information" used by the components has already been mentioned. The data structures that embody the knowledge are identified elements of the components in the second stage.

The Supervisor consists of a *task processor,* an *action sequencer,* and two data structures—the *task knowledge* and the *task status.* When a task is received from the Executive or the Operator, or when the latter reports the outcome of an action, the task processor updates the task status and determines the next action to be called. The task knowledge is used in this process. If the new action must wait for attention along with others already called, the action sequencer decides its place in the line. The task processor passes the results of tasks to the Executive.

The Operator contains an *input/output processor,* a set of *action processors, support units,* and *environmental data.* The input/output processor handles inputs from the environment, in the light of task status information received from the Supervisor, and recognizes tasks, as well as task-related data to be added to the store of environmental data. The processor also issues output to the environment as directed by the Supervisor. The action processors execute actions, using environmental data and task status information; outcomes are reported back to the Supervisor. The support units make their resources available to the processors, manage the environmental data, and do low-level scheduling when necessary.

The Executive has a *planning unit* and *decision information.* The planning unit can express a novel job in terms of

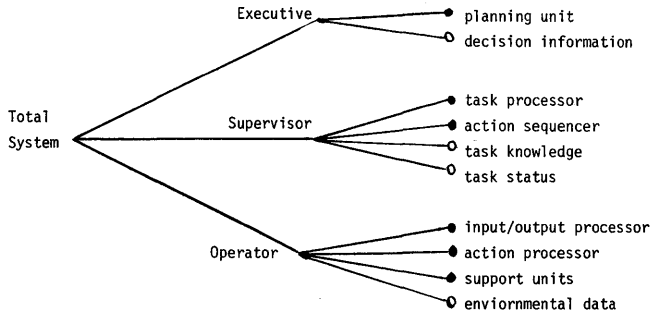


Figure 3—The structural decomposition

tasks that the Supervisor is able to manage, and it directs the sequence of such tasks, calling each new one in light of the result obtained in the current one. In the process it uses its decision information, which includes measures of performance objectives and summaries of past experience. A task result may also indicate that rules in use by the Operator need modification, or even that some Operator element should be replaced, and the Executive will then issue the necessary change.

The diagram in Figure 3 summarizes the structural decomposition arrived at in the second stage. Solid dots represent active modules, and open circles represent information elements. Active modules may themselves be constructed by applying structural decompositions to them.

Actions

As mentioned earlier, the concept of action is central to the approach being discussed. An action is an atomic unit of activity that occurs in many tasks. It is executed when it has been supplied with the resources and information it requires, and it continues until it reaches some appropriate termination point. (This is in contrast to the notion of process, used in many software systems. A process can be interrupted at arbitrary points, and resumed later. The interruption incurs an overhead cost in storing the process state and restoring it when it resumes.)

In effect, actions are the operations in terms of which tasks are programmed. The choice of the set of actions for a system is a fundamental part of the effort of constructing the system. An action should take a reasonable number of the basic clock cycles of the system. This is because each action initiation involves the Supervisor, and also requires that resources and information be supplied; the longer the duration of the action, the smaller is the ratio of system overhead to task execution. On the other hand, no action should be allowed to hold on to its resources so long that other requirements of the system are affected adversely, and this condition will set a limit to the duration of any action.

Actions should be general, so that they can respond to the variations in the environment. The Supervisor, when it calls for an action, supplies the parameters that specialize the action to the current situation.

The example in Table I, representing a very small part of the task knowledge needed by a robot chauffeur, serves to

TABLE I.—Subtask: ENTER AT (entry point) AND STOP AT (type of stopping place)

Action	Parameters	Outcome	Next Action
seek	entry point type	not yet found	seek (entry point)
		found	enter
		blocked	(terminate)
	stopping place type	not yet found	seek (stopping place)
		found	park
		not available	(terminate)
enter	entry point	not yet done	enter
		done	park
		accident	(terminate)
park	stopping place	not yet done	park
		done	(terminate)

show how a subtask might be expressed in terms of actions; and how the actions are particularized by parameters. The subtask calls for the robot to enter some specified type of location—parking lot, service station, driveway and so forth—arrived at in a previous subtask; after entering, it is to park in an appropriate spot.

The action *seek* requires that some place in the immediate environment be found that matches the type description given in the task knowledge. Therefore, the action is executed by the I/O module rather than some other processor. If the time allocated for the action is insufficient for it to find the place, the action *seek* may be continued. The subtask is terminated when the car has been parked, or when completion has become impossible. In either case, a new subtask will be determined by the Supervisor.

EXAMPLES

The application of the interactive system model to the design of a robot chauffeur has been outlined above in general terms; for further details, see Reference 7. The model has also been applied to the development of a timesharing operating system,[8] and a brief description of this application will be given.

An operating system

The environment of the operating system, as mentioned earlier, are users at terminals, and other peripheral devices. The tasks are job requests issued by users. The environmental data are user program and data files—both temporary and permanent. Examples of actions, in addition to those given earlier, are running as much of a user program as can be completed within an allowed time quantum, and inputting and outputting lines of text at terminals.

System functions are handled by the support units of the Operating component (called the Interface in the reference cited). These functions include managing core memory; allocating I/O channels, buffers, and similar resources to actions that require them; and scheduling processors to execute actions.

The Supervisor at any time is controlling the carrying out of each user's current request. When it calls an action, the call is placed on an action queue, according to its priority, by an action sequencer. The action call waits until it has received the processor, core memory, and other resources needed for its execution. The action program then carries out the action, reports the outcome to the Supervisor, and releases resources that are no longer needed. The Executive (called the Policy component in this application) can modify scheduling and other system rules to adjust to workload changes. The system administrator, who is treated as part of the Executive, can plan the handling of unusual jobs.

To illustrate the task knowledge for such a system, consider the subtask *Open* (file, access) which can be used to open a file. The following actions comprise the subtask:

transfer (file part, buffer)—transfers the next portion of the indicated file into a buffer allocated by the system.
search (buffer, file descriptor, access)—searches the buffer area to find a file descriptor. If present it determines whether the user has the indicated access right.
output (message)—prints the indicated message on user's terminal.

A representation of a simplified version of the subtask in terms of the above actions is described in Table II; the initial action is *transfer*.

TABLE II.—Subtask Open (file, access)

Current Action	Parameters	Outcome	Next Action
transfer	next portion of directory, buffer	transferred	search (file, access, buffer)
search	file, access, buffer	found and access okay	output ('OPENED')
		found and access not okay	output ('ILLEGAL ACCESS')
		not found	transfer (next portion of directory, buffer)
		EOF	output ('FILE NOT PRESENT')
output	message	outputted	—

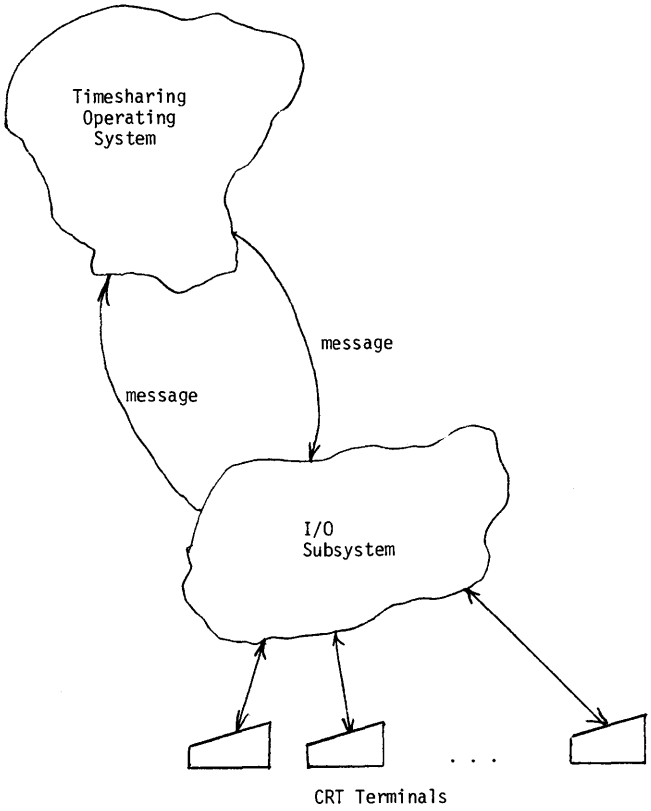


Figure 4—The I/O subsystem

A microprocessor system

As another example of the use of the model, a microprocessor system will be discussed (a more general microprocessor system is described in Reference 9). It is believed that the proposed model with its attendant concept of actions allows a methodology for the integration of hardware and software in a natural way. One example of this is in the realization of actions—an action may be dedicated to one microprocessor (or specialized piece of hardware), or several similar actions may be assigned to one microprocessor.

Consider the I/O terminal subsystem in Figure 4 that has been factored from a timesharing operating system in order to meet certain performance objectives (e.g., efficiency). Lines of input from CRT terminals are considered as messages to be transmitted between the operating system and the I/O subsystem. The subsystem itself can be decomposed applying the model. The Supervisor for such a system would take signals from the terminal, considering them as tasks, and would translate them into actions which would be executed by the Operator component. An Executive component could conceivably be used to regulate I/O traffic.

Many designs and implementations exist for such a subsystem along with their attendant opportunities for parallelism. A simplified hardware/software realization of the I/O subsystem using microprocessors is given in Figure 5. It has been assumed that the number of terminals is so large that it is infeasible to have the I/O subsystem share a processor with other parts of the system. Here there are only two components of the model—the Supervisor and the Operator. The Supervisor has a dedicated microprocessor since it must handle requests from terminals as well as outcomes of action executions. Each action has a dedicated microprocessor since many terminals may be requesting service at the same time. Thus at any given time each action microprocessor can be handling different parts of tasks issued from several terminals and the operating system. Other elements of the configuration include a decoder/router for routing action calls from the Supervisor to their respective action queues, an I/O bus and a random access memory.

The I/O bus requires two message formats—the I/O Supervisor format (see Figure 6) and the random access memory format. There is one sub-format for tasks and one sub-format for outcomes. Tasks are issued by the timesharing operating system or by a CRT terminal, while outcomes are the results of action executions.

During regular operation, terminals signal the system by placing an *Input* task on the I/O bus in Supervisor format. The task is recognized and placed on the Supervisor queue. When the task gets to the head of the queue, the Supervisor issues the first action call in the action sequence that comprises the task (see Table III).

In this case, the first action is *transfer* which transfers a message (synchronously) from the specified terminal to an internal buffer. The next action *check* determines whether the message is a command or a data item. If it is a legal command, it is *sent* to the operating system for execution, if not *transfer* outputs an error message to the appropriate terminal. If a data item is encountered, then *add* makes it the next line of the user's workspace.

The Supervisor determines the next action for a task on the basis of the outcome of the present action for this task. At any point in time an action microprocessor can be processing an action call obtained from the head of its action queue. When finished, the outcome is placed on the I/O bus

TABLE III.—Input (terminal)

Current Action	Parameters	Outcome	Next Action
transfer	terminal, action	transferred	check (buffer)
check	buffer	command	send (buffer)
		illegal command	transfer ('ILLEGAL COMMAND', terminal)
		data	add (buffer, workspace)
send	buffer	sent	
transfer	buffer, workspace	added	
		overflow	transfer ('WORKSPACE OVERFLOW', terminal)

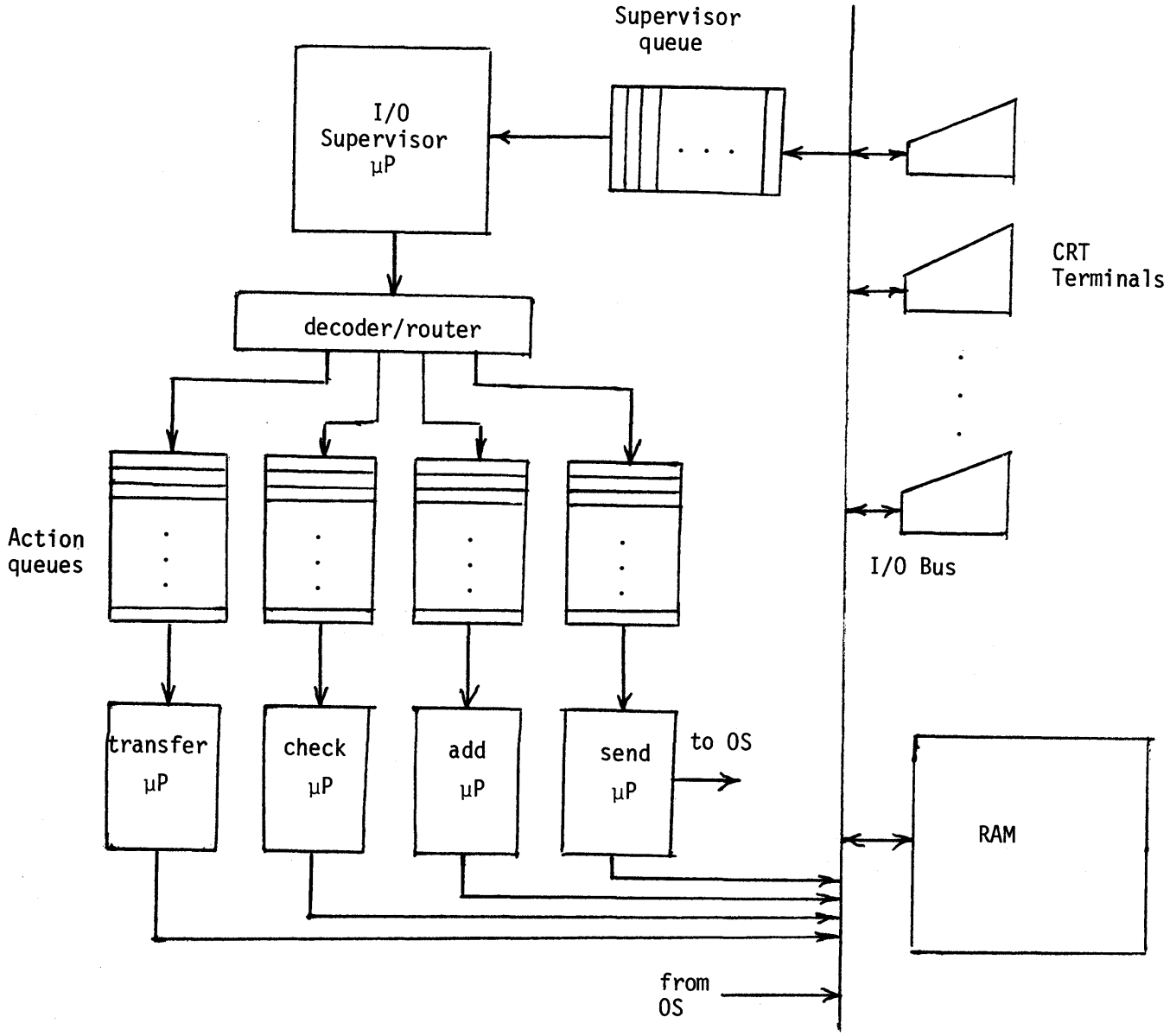


Figure 5—A hardware/software realization

and a new action call is obtained from the associated action queue. The system recognizing that the information on the bus is an outcome, puts it on the Supervisor queue. Eventually the Supervisor removes this outcome from its queue, and examines the associated task data structure to determine which action call to issue next.

0	task code	terminal no.

1	outcome code	terminal no.

Figure 6—I/O bus sub-formats

The action *transfer* is a bit different from the others. It essentially initiates an I/O operation; when the operation is complete, the associated terminal places the outcome on the I/O bus. The Supervisor of the operating system can forward a task to the I/O subsystem in an analogous way by placing a call on the I/O bus. The outcome can be returned to the Supervisor of the operating system through the *send* action.

A multi-language processor

As a final example, the application of the model to the design of a processor which executes a user's program written in an arbitrary combination of programming languages will be discussed. The model provides a natural framework

in which to realize such a processor. As a prelude, the utility of a multi-language system will be elaborated on.

Being able to write a program in a programming language suitable to expressing the solution of a problem in a straightforward manner is very desirable. String processing algorithms are easily written in SNOBOL, algebraic and vector computations are easily represented in APL, and input/output operations are most effectively specified in PL/I. Since a universal programming language which can do all of these things well is impractical at this time, it is desirable to be able to use statements from a variety of languages to express a given algorithm.

The usefulness of this approach is particularly evident in the following program segment which contains PL/I and APL statements:

```
IF x¬=0 THEN Y←+/Z1×W;
        ELSE Y←+/Z2×W;
PUT DATA (Y);
```

When Z_1, Z_2 and W are conformable matrices, even this simple two line segment is difficult to program in either PL/I or APL. It is difficult in PL/I because (1) vector operations are not available, (2) PL/I subroutines for APL operations require a different syntax, and (3) PL/I subroutines do not return arrays as values. It is difficult in APL because the equivalent of PUT DATA naming each element would require the user to generate the names of each element.

The proposed model offers a means to realize such an approach in a natural way. Consider an interactive timesharing system with a subsystem able to handle interpretively user programs written in arbitrary combinations of programming languages. Here the operating system is used to create programs, store and retrieve files, provide I/O capabilities, etc. The subsystem is invoked as an action *interpret* applied to a user program.

The subsystem consists of a Supervisor and an Operator. The Supervisor calls upon actions executed by the Operator which (1) provide a generalized fetch-execute cycle for several language processors, and (2) directly execute [10] statements in these languages. Here, then, operations such as add and compare are no longer the basic units of computation; statements in specific languages are the basic units of computation. Thus, given an assignment statement

```
X=A*B+C/D
```

in a program file which is tagged with some language specific information (e.g., language type), an action sequence controlled by the Supervisor gets the statement from the program file, identifies the indicated language, computes the expression and performs the assignment.

Variables and data structures must be specified in such a way that the meaning of each statement is unambiguous, even if it may be interpreted in more than one language. Each action can have more than one outcome. If an action associated with a statement in a particular language executes properly, it may, for example, assign a value to a variable or indicate the next statement to be executed. Otherwise, it may fail to parse or encounter execution difficulty; either of these will cause a return to the operating system.

CONCLUSION

This paper has presented a model and methodology for the construction of interactive systems. Its usefulness has been shown in its application to realizing timesharing operating systems, robot-chauffeured automobiles, microprocessor systems and multi-language processor systems. It is felt that a structural approach of this nature is necessary in managing the complexity of the decomposition of large interactive systems.

REFERENCES

1. Dijkstra, E. W., "The Structure of the T.H.E. Multiprogramming System," *CACM*, Vol. 11, No. 5, May 1968, pp. 341-346.
2. Mills, H., "Top Down Programming in Large Systems," in *Debugging Techniques in Large Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, 1971.
3. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *CACM* 15, No. 12, December, 1972, pp. 1053-1058.
4. Jacobs, W. W., "A Structure for Systems that Plan Abstractly," *AFIPS Conf. Proc.*, Vol. 38, 1971, pp. 357-364.
5. Jacobs, W. W., "Control Systems in Robots," *Proc. ACM 25th Anniv. Conf.*, Vol. 1, 1972, pp. 110-117.
6. Jacobs, W. W., "How a Bug's Mind Works," *Cybernetics, Artificial Intelligence, and Ecology*, Spartan Press, 1972.
7. Jacobs, W. W., "The Robot Chauffeur," *Proceedings of the Third Milwaukee Symposium on Automatic Computation and Control*, 1975.
8. Freeman, M., W. W. Jacobs, and L. S. Levy, "A Model for the Construction of Operating Systems," (in preparation).
9. Freeman, M., W. W. Jacobs, and L. S. Levy, "On the Construction of Microprocessor-Oriented Operating Systems," *Proceedings of the 1977 International Conference on Parallel Processing* (to appear), Shanty Creek, Mich., August, 1977.
10. Chu, Y., *High-Level Language Computer Architecture*, Academic Press, New York, 1975.

Software fault-tolerance in the Pluribus

by JOHN G. ROBINSON and ERIC S. ROBERTS
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

Over the past decade, the decreasing cost of minicomputer components has encouraged the use of multiprocessor techniques in the design of high-speed, cost-effective computer systems. Multiprocessor architectures have two principal advantages over conventional single-processor designs. First, a multiprocessor system can achieve greater computational speed through parallelism in its task structure. A second advantage is that multiprocessors realize greater reliability through redundancy of processing elements. This paper discusses several aspects of multiprocessor reliability as it applies to the Bolt Beranek and Newman (BBN) Pluribus system.[1]

Recent advances in processor technology, including the development of inexpensive LSI-based microprocessors, have further increased the cost advantages of multiprocessor systems and have led to the consideration of multiprocessor architectures involving very large numbers of processors. Early examples of multiprocessor systems designed around minicomputers include the C.mmp system at Carnegie-Mellon[2,3] and the BBN Pluribus. Each of these early systems has prompted the development of larger microprocessor-based systems, such as the Carnegie-Mellon Cm* project[4] and the BBN "Butterfly" machine.

THE BBN PLURIBUS

The BBN Pluribus system has been described in detail in a number of previous papers.[1,6] The Pluribus was originally designed as a reliable, modular, high-speed Interface Message Processor (IMP) for use with the ARPANET.[7] To date, eight such systems have been delivered. The Pluribus has also been used to support a real-time processing and display system for seismic data[8] and a terminal handler connecting a large number of terminals to several host computers.[9] Future applications include a high-speed version of the Satellite IMP.[10] On the whole, our experience with building Pluribus systems has been favorable, and we are currently trying to attract a more general market.

The basic organization of a typical Pluribus system is illustrated in Figure 1. Each processor bus contains a number of processing units (usually two) and associated local memory for each processor. The processors are SUE minicomputers produced by the Lockheed Electronics Corporation. Common memory (that which is accessible to all processors) is distributed among the system memory busses. The I/O bus units provide an interface to a variety of input/output devices, such as communication lines and modems in conventional Pluribus applications. In addition, each I/O bus includes a real-time clock used to coordinate processor activity and a special hardware device called a PID (Pseudo Interrupt Device) used for process scheduling. The individual busses are connected to each other via bus couplers which are indicated by the solid lines in Figure 1.

The hardware organization of the Pluribus makes it possible to achieve system reliability through the redundancy of hardware components and interconnection paths. For example, if a processor bus becomes unusable through hardware failure, the remaining processor bus(ses) will generally be able to provide sufficient computational power to run the system. Similarly, if the connection between a processor and common memory bus is lost, the system can proceed normally by removing either of the two busses from the operational system.

RELIABILITY IN THE PLURIBUS

Fault-tolerant computer techniques originally arose in response to the unreliability of early digital hardware.[11] Historically, the principal concern of fault-tolerant system design has been with completely error-free system operation. The classical technique used to provide reliability at this level is to include three or more instances of each system component; each component performs each calculation simultaneously. Additional hardware is then used to check the individual results for correctness, relying on a majority decision in the case of a discrepancy.

The notion of relaxed reliability

Most of the past research in reliable computer system architecture has been directed toward applications in which completely fault-free operation has been necessary. In designing the Pluribus, we have been concerned with applications whose reliability requirements are much less stringent. We have found that these applications suggest a

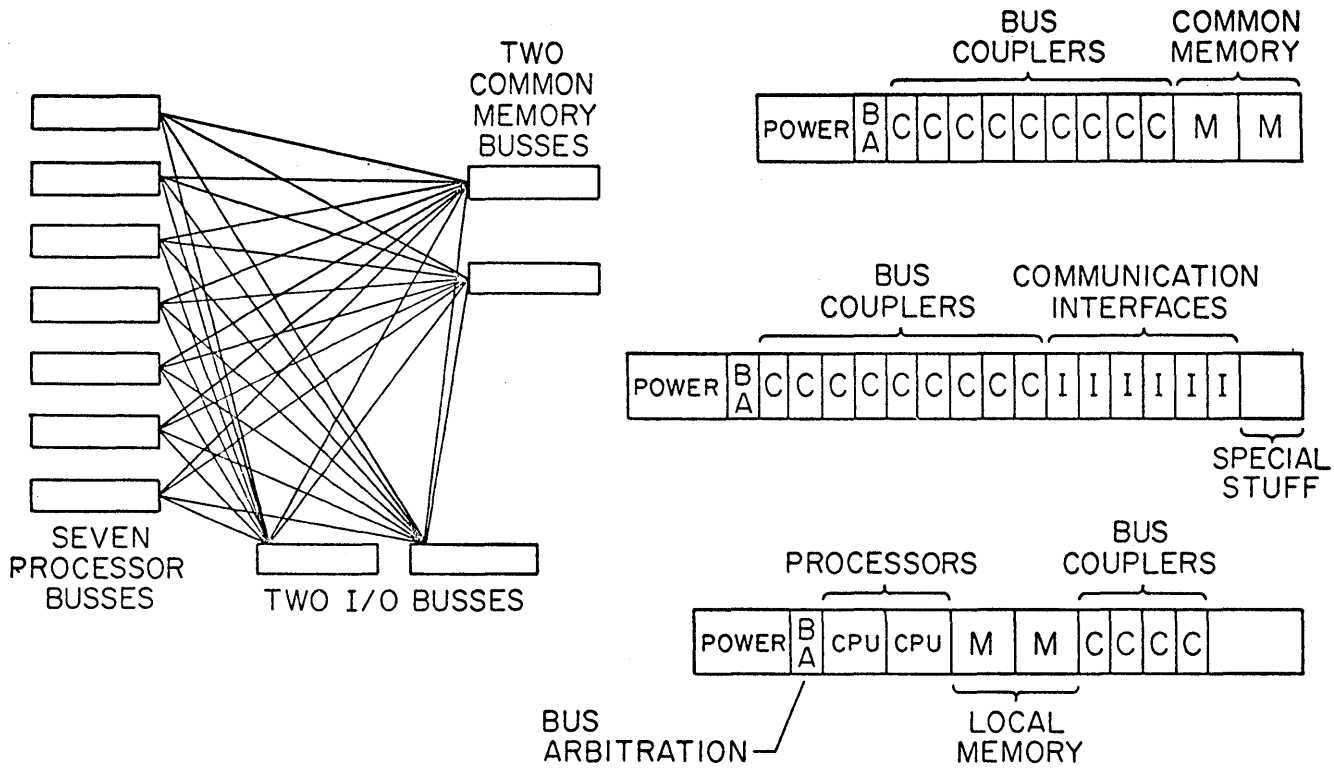


Figure 1—Typical Pluribus System Configuration

relatively new set of techniques and that many of the classical approaches to fault-free reliability do not directly apply.

In the Pluribus, we have not been as concerned with continual error-free operation as with providing maximum system availability. In the environment of a communications network, for example, it is desirable to keep each node functioning even if data is lost during a failure-induced transient. A communications network is characterized by a relatively steady-state operation which is largely independent of all but the most recent network activity. If a node in the network is forced to reinitialize, some active message traffic will generally be lost, but the node will very quickly be able to handle new messages in a normal fashion. This property of the communications network makes a relaxed reliability mechanism very attractive in that the continuity of system operation can be insured without incurring the cost of a more elaborate hardware reliability scheme.

We feel that the relaxed reliability model is widely applicable in other areas. For example, by enhancing a general-purpose operating system with checkpointing facilities, it should be possible to create an environment where such approaches to fault-tolerance may be used to advantage. The techniques used in the Pluribus should permit such a system to reconfigure following a failure and automatically resume normal operation from the previous checkpoint. In the network environment, checkpointing is not necessary since there is no long-term context to be saved. A system with checkpointing approximates the network environment by providing a clean state (the previous checkpoint) at which operation can resume.

Relaxed reliability in the Pluribus

Unlike most conventional systems, the principal responsibility for maintaining reliability in the Pluribus is placed on the system software rather than in the hardware structure. The Pluribus hardware was designed to provide an appropriate vehicle for the software reliability mechanism. When hardware errors are detected, the software exploits the redundancy of the hardware by constructing a new logical system configuration which excludes the failing resource, using redundant counterparts in its place.

Pluribus systems make use of redundancy in their software structures for much the same reason. Redundant information is intentionally introduced into the data structures at various points and checked by processes operating upon those structures. An example of this technique applied to buffer structures is described later. In addition, periodic background processes are used to recompute certain variables which are maintained by the operational system. If the recomputation uncovers a discrepancy, the variables are fixed or a more drastic recovery is attempted.

In many cases, a failure is not detected at the exact time of occurrence but at the time that the software encounters some failure-induced discrepancy. By this time, the effects of the failure may be more widespread and the actual cause of the failure may be difficult to detect. In such cases, the system is not able to perform instantaneous recovery and seeks instead to restore normal operation as quickly as possible.

The value of software reliability

Our experience with the Pluribus has convinced us that fault-tolerance need not be expensive, provided complete fault-free operation is not the goal. Pluribus hardware can be configured for redundancy by including one extra copy of each critical system resource (i.e., one extra processor bus, one extra memory bus, and so forth). In general, software reliability mechanisms are highly flexible and are largely independent of the application routines.

As a result of placing the responsibility for system recovery on the software, Pluribus systems are capable of surviving (or isolating the effects of) a wide variety of intermittent failures and design errors in either hardware or software. Diagnosis of the failures can often be accomplished while normal operations continue.

THE PLURIBUS OPERATING SYSTEM

This section discusses the organization of the Pluribus operating system and some of the techniques used for achieving coordination of multiple processors. These techniques are further explored in later sections, which provide two examples of Pluribus fault-tolerant software strategies. One of these examines the Pluribus IMP buffer system in detail, and the other covers strategies for understanding failures when they occur and effecting necessary repairs.

General responsibility of the operating system

The software reliability mechanisms for a Pluribus system are coordinated by a small operating system (called "Stage") which performs the management of the system configuration and the recovery functions. The overall goal of the operating system is to maintain a reliable, current map of the available hardware and software resources. The map must include accurate information not only about the hardware structure of the machine, but also about variables and data structures associated with the processes that use that hardware. Moreover, the operating system must function correctly even after parts of the system hardware have ceased to be operational. New resources, as they are discovered (e.g., because hardware has been added or repaired), should be incorporated as part of the ongoing operation of the application system.

Since any component of the system may fail at any time, the operating system must monitor its own behavior as well as that of the application system. It may not assume that any element of hardware or software is working properly—each must be tested before it is used and retested periodically to ensure that it continues to function correctly. The operating system must be skeptical of its current picture of the system configuration and continually check to see if the environment has changed.

The Pluribus operating system builds the map of its environment step by step. Each step tests and certifies the proper operation of some aspect of the environment, relying on those resources certified by previous steps as primitives. Early steps examine the operation of the local processor and its associated private resources. Subsequent steps look outward and begin to discover and test more global resources of the system, giving the checking process a layered appearance. In the Pluribus operating system, each processor begins by checking its own operation and by finding a clock for use as a time base. Once these resources have been verified, the processor can begin to coordinate with the other active processors to develop an accurate picture of the system.

At the same time, the system must balance the need for reliable primitives with the need to accomplish normal operation efficiently. When all the environment has been certified, the system should spend most of its processing power on advancing the operational algorithms and return only occasionally to the task of reverifying its primitives. When failures of the environment have been detected, however, the power of the system must be brought to bear on the task of reconfiguring to isolate the failure.

Hierarchical structure of the stage system

The Pluribus operating system is organized as a sequence of stages which are pooled by a central dispatcher. A processor starts with only the first stage enabled. As each stage succeeds in establishing a proper map of its segment of the system state, it enables the next stage to run. Each stage may use information guaranteed by earlier stages and thus may run only if the previous stage has successfully completed its checks. Once enabled, a stage will be polled periodically to verify that the conditions for successful completion of that stage continue to apply. The system applies most of its processing power to the last stage that is enabled but returns periodically to poll each earlier stage. The application system is the final stage in the sequence and may run only after the earlier stages have verified all the config-

TABLE I.—Pluribus Operating System Stages

STAGE	FUNCTION
0	Checksum local memory code (for stages 0, 1, 2). Initialize local interrupt vectors, and enable interrupts. Discover Processor bus I/O. Find some real-time clock for system timing.
1	Discover all usable common memory pages. Establish page for communication between processors.
2	Find and checksum common memory code (for stages 3, 4, 5). Checksum whole page ("reliability page").
3	Discover all common busses, PIDs, and real-time clocks.
4	Discover all processor bus couplers and processors.
5	Verify checksum (from stage 2) of reliability page code (for rest of stages plus perhaps some application routines). External reloading of missing code pages is possible once this stage is running.
6	Checksum all of local code.
7	Checksum common memory code. Maintain page allocation map.
8	Discover common I/O interfaces.
9	Poll application-dependent reliability and initialization routines. Periodically trigger restarts of halted processors.
10	Application system.

uration information of the application and the validity of the data structures.

Table I lists each stage of the Pluribus operating system, together with the aspects of the environment it guarantees. Many of the functions listed will not be discussed further but are provided to illustrate the layering of stages.

Since processors continue to perform each of the stages periodically, changes in the environment will eventually be noted. Any stage detecting a discrepancy in the configuration map will disable all later stages until the discrepancy is repaired. Then, all the later stages, which might depend on data verified by the disabling stage, will be forced to run all their checks, guaranteeing that they will make any further modifications to the configuration map necessitated by the first change. A serious failure, such as a non-existent memory interrupt, disables all but the first stage. In these cases, some reconfiguration might be needed, and all stages should perform all their checks before the application system is resumed.

Establishing communication

So far, we have described the progress of one processor through the staged checking procedures of the operating system. All processors in the Pluribus perform the same checks, since it is important that they agree about the state of the system resources. Coordination of multiple processors with potentially different views of the hardware configuration requires two mechanisms: the processors must agree on an area of common memory in which to record the machine configuration map, and they must cooperate in their decisions to modify the map.

The first step in coordinating the multiple processors of a Pluribus is to agree on a page of memory through which to communicate. The procedure for initially establishing the page for communication is clearly delicate. Prior to establishing the page, the processors have no way to communicate about where it will be. The procedure must operate correctly in the face of failures which might leave some of the processors seeing a different set of common memory pages from the rest. Processors which are unable to see the communication area will attempt to use another memory page and must be prevented from interfering with the unaffected processors.

Any processor that is first starting up (or restarting after some massive failure) can assume nothing about the location of the communication page. Any page may be used, and therefore a small area for communication control variables is reserved on each page of common memory. Part of this area is used for a brief memory test, which must succeed before the page may be used at all. Every processor attempts to establish the lowest-numbered (lowest address in memory space) page that it sees as the page through which to communicate. To be valid, any page must have a pointer to the current communication page, and the communication page must point to itself.

Each processor looks at the pointer on the lowest-numbered page it can see. There are three possible states for the pointer. First, if it points to the page itself, the processor has found the communication page and may now proceed to interact with other processors about the common environment. If it points to a higher-numbered page, the processor may just fix the pointer, as the requirement that the communication page be lowest makes this case inconsistent. If it points to a lower-numbered page, the processor must attempt to check if the indicated communication page is active. It must assume that the data might simply be old or invalid and must time it out using a dedicated entry in a special array of timers which is allocated on each page. The processor increments the timer, and, if it ever reaches a certain threshold, unilaterally fixes the communication pointer, and starts to use this page for communication. The processor is prevented from doing this by any other processor which is successfully using the lower-numbered communication page; such a processor will periodically zero all the timers in the array for each memory page in the system before the threshold is reached.

Consider what happens during various possible hardware failures. If the memory bus containing the communication page is lost, all processors will attempt to establish a new communication page on the other bus. Using their timers on the new lowest page (which initially points to the old one after the failure), they await the threshold. No one is holding the timers to zero, so the new page becomes the communication page when some processor's timer first runs out.

A processor blinded to the communication page by a bus or coupler failure will try to establish a higher-numbered page for communication. From the point of view of the failing processor, this case is indistinguishable from the previous case, where the common bus failed. Since the rest of the processors are satisfied with the communication pointer, they will hold all timers to zero, and the failed processor will never be able to change the communication page pointer. If the processor sees a set of pages disjoint from the rest of the system, it behaves as if no other processors are running, but there is no memory where it may interfere and now we have two systems operating independently. In this case it is likely that the two systems will interfere over other resources; since multiple failures are required for this situation to occur in a Pluribus, we choose not to attempt recovery here.

The consensus mechanism

When configuration data must be updated, it is crucial to coordinate the Pluribus processors before making the modification. The mechanism to accomplish this goal we call consensus. Each stage has a consensus which is maintained as part of its environment. The first step in forming a consensus is to determine the set of processors that is executing the corresponding stage. This set has certified the primitives necessary to maintain successfully this stage's portion of the configuration map. In order for the system to respond to failures, the consensus must be kept current—new processors must be able to join it rapidly and processors that may

have halted or ceased to run the stage must be erased from the set.

Each processor, based on its hardware address in the Pluribus, is assigned a bit in three consensus arrays, called "next," "smoothed," and "fix-it". As part of running the corresponding stage, every processor periodically sets its bit in the next consensus array to show that it wishes to participate in the consensus. After enough time has elapsed for each properly running processor to set its bit, this array is copied into the smoothed consensus and cleared. The set of processors in the smoothed array will then be used as a basis for decisions to reconfigure some portion of the resource map.

Any processor which wishes to modify some configuration information sets its bit in the appropriate fix-it array. Processors that agree with the configuration map clear their bits, and bits corresponding to processors not in the smoothed array are also cleared.

In effect, the bits in the fix-it array represent the votes of the individual processors in favor of a potential modification. In most cases, it is desirable that all processors agree before making the change. All processors wait until the fix-it array matches the smoothed array before implementing the fix. Other modifications might require only majority or two-thirds agreement. The choice of policy often depends on some trade-off between resources (e.g., should we use more memory or more processors?). The Pluribus approach allows us to make this choice independently at each stage.

Since each processor in the Pluribus performs each stage of the checking code, the consensus mechanism provides the coordination needed to change the configuration map gracefully. As one of its stages detects a failure, the processor sets the appropriate fix-it bit and disables the following stages. When enough processors detect the failure they implement the fix to the configuration map. Now these processors can complete the later stages, devoting their attention to any further changes required by the failure. A processor which sees a different picture of the resources and cannot reach agreement with the rest of the system hangs forever at the point of detecting the discrepancy. This technique effectively prevents the processor from damaging the system.

Application-dependent checking

In general, it is desirable for the application system to perform its own checks before initiating or resuming normal operation. The last stage provides a mechanism which polls application-oriented processes to perform consensus-driven checks and repairs of their own data structures. This stage uses the results of the hardware (application-independent) discovery stages to certify its own data structures. For example, it could allocate or deallocate device parameter blocks as the devices are discovered or disappear and initialize spare memory pages for use as data buffers as they become available. User-written reliability checks can be performed on any of the application data structures, and the appropriate reinitialization invoked to remedy failures.

Occasionally, it is possible for a processor checking application data structures to implement minor repairs to the data structures unilaterally. For major reconfigurations of the data structures, such as complete application system reinitialization, the checking routines must signal to the stage dispatcher that consensus is needed. The last concurring processor is then permitted to perform the reinitialization routine. Just as the early stages guarantee the hardware map, the application-dependent routines have the consensus mechanism at their disposal to validate the system data structures before entering the system. In addition, the application system data structures are rechecked periodically during normal system operation.

AN EXAMPLE OF APPLICATION RELIABILITY

We use two general techniques to ensure the validity of data structures in the Pluribus. First, redundant information, where it exists, is checked for discrepancies, and appropriate action taken if they exist. Second, since detailed examination of all data for inconsistency is deemed impossible for any system of non-trivial complexity, we use watch-dog timers to ensure the correct operation of the application system at various levels. As an example, we will discuss the buffer management strategy for the Pluribus IMP system.

Buffers in the Pluribus IMP circulate through the system from queue to queue; in some cases, they may be shared between two or more processes. Since a compromised queue structure may, in general, rapidly degrade the performance of the system, elaborate checking methods are built into the IMP program at various levels. In particular, we must be able to detect queues that are crossed or looped and buffers that have been lost (are on no queue at all).

Associated with each buffer in the system is a set of use bits corresponding to various processes that consume buffers. Any process that enqueues a buffer for some other process first sets the use bit for that process. When a process dequeues a buffer, the appropriate use bit must be on or the buffer will not be processed. As a special case, buffers on the system free list must have all their bits turned off. The buffer-freeing routine only returns a buffer to the free list if the last remaining use bit is that of the freeing process.

This technique intentionally generates redundant information and continually validates it as a buffer circulates through the system. In other words, the existence of a buffer on a queue informs the system that some processing is desired for that buffer. In principle, the use bit signals the same thing. Each buffer-processing routine could scan all the buffers in the system for those with its use bit set, but such strategy would clearly be inefficient. The redundancy check gives preference to neither the queue nor the use bit as an indication of need for service, but rather requires agreement between the two indicators. When they disagree, the system assumes that a failure has indeed occurred and attempts to correct it by forcing the queue to be empty, so that the effects of the failure can be contained as much as possible.

The use bits allow the prompt detection of looped and

crossed queues. In addition, an improper buffer pointer will often lead to a failure of the use bit check. We must also consider the case of a buffer which has been lost from all queues. This condition could arise due to a program bug or as a result of a queue being emptied after a use bit failure. We could employ a classical garbage-collection scheme for this purpose; unfortunately, the demand on buffers is often large in a high-speed communication system, and the requisite locking of the buffer resources during such a garbage collection would likely result in lost inputs.

The recovery scheme we have chosen is a watch-dog timer mechanism. Each buffer has associated with it a flag which is set by normal activity of the buffer, which in this case is defined to be the periodic appearance of that buffer on the free list. Whenever a buffer is freed, its flag is set. In addition, flags for all the buffers on the free list are set periodically. In the high-speed communications environment, where data passes through a network node very rapidly, each buffer must appear on the free list at least once every two minutes. Therefore, each buffer flag is checked every two minutes to be sure it is set, and then cleared. A zero flag indicates that the buffer has dropped out of normal activity, and the buffer is unilaterally freed and its use bits cleared. In this way, any lost buffer is detected within at most four minutes and returned to normal usage.

FAILURE DIAGNOSIS IN THE OPERATIONAL SYSTEM

The Pluribus design permits the continuation, after perhaps a brief outage, of normal system operation following the failure of any single component. The component might be a single bit of memory or an entire common memory bus. Much of the responsibility for ensuring recovery from failures lies in the software; very little hardware beyond that necessary to construct a multiprocessor should be required.

The initial Pluribus systems performed their recovery quite well in many cases. Minor problems were often repaired so effectively that the maintainers and users were never aware of the repair being carried out. Even following drastic failures, such as the loss of a common memory bus, normal system operation was restored within seconds. As we gained experience with the operational systems, however, certain deficiencies in our strategies became clear.

In some failure cases, one repair would lead to another, until eventually a fairly major reinitialization would be performed, with obvious effects on the users of the system. Unfortunately, the massive recovery often destroyed the evidence of the original failure, or occurred too late to permit effective diagnosis. While the stated goal of restoring the system to normal operation was achieved, we were left without any idea of why the reinitialization was necessary. This was particularly frustrating when the frequency of the reinitialization was on the order of hours or days.

In other cases, normal operation seemed to continue, while some hardware failure occurred undetected. Either the failure was covered by effective recovery at a fairly low level in the system or it occurred in a redundant portion of the hardware which was not being exercised. A second failure, in conjunction with the first, would remove the last copy of some critical resource, causing the system to fail.

Such experiences have led us to an improved design for the recovery software. Any recovery operation is reported by the system using a system trap (i.e., a supervisor call). Even in those cases where an error might occur occasionally in normal operation, we prefer to report the failure and filter out the non-important occurrences of the report. In this way, an incipient hardware failure is often detected early enough that it can be remedied well before the inevitable second failure brings the system down.

We have also adopted strategies which exercise all the hardware in the machine periodically. Including a spare copy of some resource helps the system recover only if that spare resource is known to be good. Therefore, we force the system to exercise all of its resources from time to time. In some cases we use manual procedures, but the tendency has been to include automatic rotation procedures in the operational system software.

One beneficial side-effect of this policy is that the operational program has become the best diagnostic for the hardware. Traditionally, some of the most subtle hardware failures occur during operation of the application system, though the hardware diagnostic program never detects any errors. By augmenting the operational system with diagnostic capabilities, we are able to isolate hard-to-find or intermittent failures, sometimes even without interrupting normal operation. The system-integration personnel routinely use the operational program as the last step in constructing a new Pluribus machine.

Understanding the nature of a failure in the running system requires fairly accurate knowledge of the state of the machine at the instant of the failure. The initial implementation of the trap mechanism recorded only the code number of the trap, which processor or set of processors had encountered it, and a total occurrence count. More recently, we have augmented the trap mechanism to allow for saving a larger snapshot of the instantaneous state of the processor, including such information as the contents of the general registers, the global system time, map register settings, the last value read from the PID, and other important local data. These snapshots allow us to examine critical information about the failure after the fact, while permitting the recovery code to take effect and normal operation of the system to proceed.

To aid in the development of new software, we have also included a password-protected facility whereby a processor which has detected a serious malfunction may signal all the other processors to stop before obtaining their next task from the PID. This capability allows examination of the failure in the global environment, which might otherwise be repaired by recovery code or cause some other more dramatic failure. These mechanisms combine to provide the programmer with a reasonably tractable debugging environment. In particular, such facilities have greatly aided software development for new Pluribus applications.

SUMMARY

The structure of the Pluribus operating system has been described above, together with some techniques for software reliability and fault diagnosis. We have built Pluribus systems for several applications, and our experience has been favorable on the whole. While the emphasis of our applications has been upon communications, we expect our techniques could be applied to many different uses. Recoverable faults arise in most computer applications, and we feel that non-redundant computer systems would benefit from the incorporation of recovery-oriented operating systems such as that of the Pluribus.

ACKNOWLEDGMENTS

Much of the initial development of the Pluribus computer was supported by the Information Processing Techniques Office of the U.S. Defense Advanced Research Projects Agency, under Contract Numbers DAHC15-69-C-0179, FO8606-73-C-0027, and FO8606-75-C-0032, and by the Defense Communication Agency under contract DCA200-C-616. Additionally, a number of the application systems were developed under contracts from various branches of the U.S. Government.

Many people have contributed to the Pluribus project; Frank Heart has led the effort since its inception. Will Crowther, Mike Kraley, and Bill Mann in particular have contributed many of the ideas incorporated into the Pluribus operating system. Jane Barnett and Bob Brooks provided encouragement and help with the actual writing of the manuscript.

REFERENCES

1. Heart, F. E., S. M. Ornstein, W. R. Crowther and W. B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings*, Vol. 42, June 1973, pp. 529-537.
2. Cohen, E., "Symmetric Multi-Mini-Processors: A Better Way to Go?" *Computer Decisions*, January 1973, pp. 16-20.
3. Wulf, W. A. and C. G. Bell, "C.mmp—A Multi-Mini-Processor," *AFIPS Conference Proceedings*, Vol. 41, pp. 765-777.
4. Swan, R. J., S. H. Fuller and D. P. Siewiorek, "Cm*—A Modular, Multi-Microprocessor," *AFIPS Conference Proceedings*, Vol. 46, June 1977, pp. 637-644.
5. Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel and F. E. Heart, "Pluribus—A Reliable Multiprocessor," *AFIPS Conference Proceedings*, Vol. 44, May 1975, pp. 551-559.
6. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 543-549.
7. Gudz, R. T., "Application of the Pluribus Multiprocessor in a Distributed Data Collection and Processing Network," *Conference Record of OCEANS 77*, October 1977.
8. Mann, W. F., S. M. Ornstein and M. F. Kraley, "A Network-Oriented Multiprocessor Front-End Handling Many Hosts and Hundreds of Terminals," *AFIPS Conference Proceedings*, Vol. 45, June 1976, pp. 533-540.
9. Butterfield, S. C., R. D. Rettberg and D. C. Walden, "The Satellite IMP for the ARPA Network," *Proceedings of the Seventh Annual Hawaii International Conference on System Sciences*, Honolulu, Hawaii, January 1974, Computer Nets Supplement, pp. 70-73.
10. Avižienis, A., "Approaches to Computer Reliability—Then and Now," *AFIPS Conference Proceedings*, Vol. 45, June 1976, pp. 401-411.

MTR—A tool for displaying the global structure of software systems

by GUY de BALBINE
Caine, Farber & Gordon, Inc.
Pasadena, California

INTRODUCTION

The purpose of this paper is to propose a new computer based method for displaying the global structure of software systems. We are particularly motivated by finding a manageable representation of the directed graph (digraph) of calls within a design or an executable program, each directed edge representing one or more calls from a procedure to another. We expect such a representation to provide a better understanding of the relationships between various components of a software system and to lead to the design of systems that are better organized.

The directed graph of calls within a program is an example of what we call invocation digraphs. Invocation digraphs are a particular class of directed graphs with at most one directed edge from one vertex to another or to itself. These invocation digraphs constantly arise during the design, implementation and maintenance of software systems.

Usually, invocation digraphs are drawn by hand if they are drawn at all. "Structure charts"[14] provide the most familiar example of a graphic representation which is visually informative. However, such a technique is of limited value in practice because of the considerable effort required to draw even a graph of modest size—and large graphs are the most interesting because they are the least understood. One of the major difficulties is deciding where to position adequate size boxes on the page so that the resulting structure becomes visually informative instead of hopelessly confusing. Ofter, a judicious positioning of the boxes becomes apparent only after the first charting is essentially complete. Even if a good layout is obtained, the charts remain cumbersome and expensive to maintain and modify.

One obvious solution is to try mechanizing the whole drafting process. This certainly alleviates the manpower problem but leaves other problems still unsolved—how to dimension and position boxes, draw edges that do not cross over at random, handle the off-page connector problem and cope with the lack of adequate graphic devices at many installations.

As an alternative, we propose an extension of the simple yet powerful indented representation of trees, capitalizing on the observation that most invocation digraphs of interest are loop free or almost loop free. We call this one dimensional, indented representation the Modular Tree Representation (MTR) of the invocation digraph.

In an MTR, the position of each node (procedure, subroutine, . . .) is chosen systematically to reflect its relationship with other nodes in the structure. Although the connections are not explicitly materialized by lines, the specific location of each node in the MTR implies to the reader that certain connections must exist. Thus, the MTR emphasizes the global structural properties of the digraph while relegating specific path connections to a secondary role. We believe this approach to be practical because invocation digraphs are most often used to grasp a morphological "understanding" of the structure of a software system rather than to trace through many individual paths.

DEFINITION OF INVOCATION GRAPHS

We start with a set of components numbered 1 through n. Components is a loosely defined term which will mean subroutines, procedures, design segments,[2] . . . , depending upon the context.

Let us construct the *invocation graph G* as follows:

- create for each component an associated vertex, e.g., v_i corresponds to the ith component;
- for all pairs of vertices (v_i, v_j) draw a directed edge from v_i to v_j if there is at least one invocation of the jth component from the ith one.

In some simple cases, the invocation graph may turn out to be a tree but, in general, we expect a directed graph, some of its vertices having indegrees greater than 1. Invocation graphs are a subset of directed graphs since there can be at most one edge between any pair of vertices, distinct or not. Notice that invocation digraphs have loops if coroutines or recursively callable components are present.

We shall use the $n \times n$ adjacency matrix A whose element A (i,j) has the value

1 if there is a directed edge from v_i to v_j
0 otherwise

The reachability matrix R is an $n \times n$ matrix such that its element R (i,j) has the value

1 if there exists a directed path, i.e., a sequence of directed edges from v_i to v_j
0 otherwise

Invocation digraphs usually have only one begin node v_1, i.e., v_1 is the only vertex in G with indegree 0. But they may have several or even none at all if every component is at least invoked once.

If several vertices with indegree 0 exist and/or the graph is not at least weakly connected with begin node v_1 (i.e., the corresponding undirected graph is not connected), we create another vertex v and draw a directed edge from v_0 to each node with indegree 0. We then determine if all vertices are reachable. If some vertices are still not reachable from v_0, we select any such vertex, connect v_0 to it with a directed edge and compute the reachability again. This process repeats until all vertices are reachable from v_0. Although this algorithm clearly forces the graph to become weakly connected (which is our requirement), it is inadequate in practice. In the appendix, we shall present some refinement to this algorithm guaranteeing that the number of vertices artificially connected to v_0 is minimal. At any rate, we can now renumber the vertices so that v_0 becomes v_1 and so forth. Thus, we have obtained a weakly connected invocation graph with begin node v_1. Without loss of generality, we can, therefore, turn our attention to weakly connected invocation digraphs with begin node v_1 only.

MTR PROPERTIES

The MTR is an extension of the common indented representation of trees. By design, the MTR becomes the preorder representation if applied to a tree. In that case, the tree is traversed from left to right, visiting descendants first. Each vertex traversed is listed on a new line, horizontally indented to show its distance from the root. An example is shown in Figure 1.

The purpose of the square bracket to the left of v_1 is to make the scope of v_1 visible, i.e., to identify the set of vertices that can only be reached by traversing v_1. Since v_1 is the root of the tree, every vertex is enclosed in the outermost and only bracket. The role of brackets will become obvious when we examine more complex MTR examples later.

In practice, invocation digraphs are rarely as simple as trees. As soon as a procedure is called from two distinct procedures, the invocation graph no longer is a tree. Due to the non-recursive nature of most software systems being built, we expect invocation graphs to be acyclic or almost acyclic.

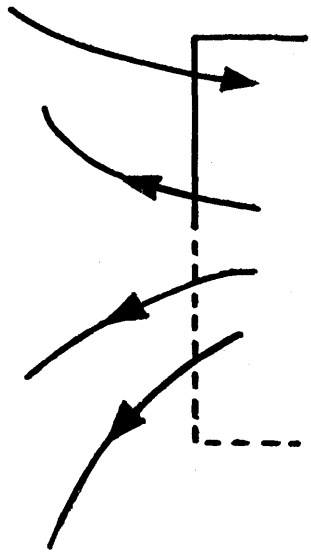
This observation is crucial for the MTR. Notice that it is simply a paraphrase of the fact that most software systems are hierarchical in nature.[3] They have a top and a bottom—high level components and low level ones. Thus, we can expect a representation that displays these hierarchical properties to be useful both at the global and local levels.

To introduce the MTR of a non-trivial digraph, let us informally examine Figure 2. Looking at part 1 of Figure 2 first, we have separated the subgraph into four disjoint trees by breaking the incoming edges to vertices with indegrees greater than 1.

The four subtrees rooted at vertices 1, 6, 9 and 11 each have an associated bracket in the MTR. These brackets allow the following "understanding" of the graph:

—1 calls 2, 3, and 4 which are called by no one else.
  2 calls 5 and 6, but 6 is called elsewhere
  . . .
—the trees rooted at 6 and 9 are referred to by the tree rooted at 1 (because they are in the broken part of the bracket associated with 1)
—the tree rooted at 9 is referred to by the tree rooted at 1 but also by the tree rooted at 6 (since it is in the broken part of the bracket associated with 1 but also indented to the right of 6)
—the tree rooted at 13 is only referred to by the tree rooted at 9 (since it is in the broken part of the bracket associated with 9)

In particular, we can infer from the MTR that the tree rooted at 13 is not known outside the "scope" of 9. Similarly, the trees rooted at 6 and 9 are not known outside the "scope" of 1. This property of not being known outside the "scope" of a particular vertex is fundamental for using the MTR. Each square bracket shields the vertices that are defined within it, except the root, from outside references, i.e., there are no incoming edges except possibly to the root. On the other hand, references from within the square bracket to outside vertices are freely permitted.



In order to arrive at this MTR, we have made use of the dominance concept. It can be summarized as follows: in a weakly connected graph with a begin node of v_1, we say

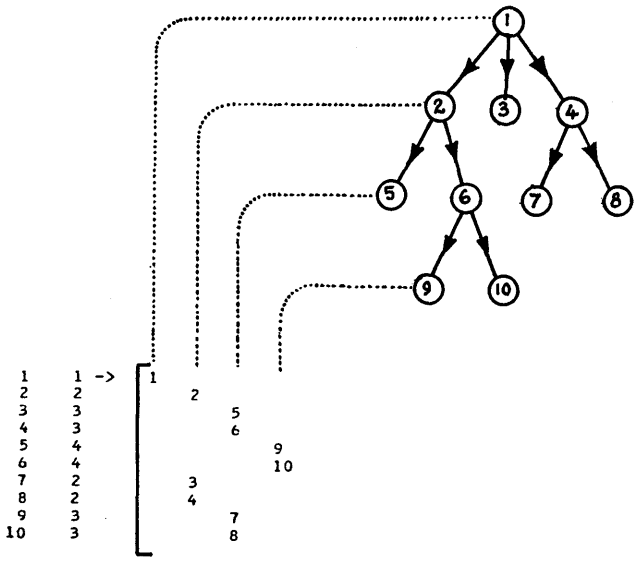


Figure 1—MTR of a tree

that v_i *dominates* v_j if $v_i \neq v_j$ and every path from the begin node v_1 to v_j contains v_i.

We say that v_i *directly dominates* v_j if:

(i) v_i dominates v_j;
(ii) if v_k dominates v_j and $v_k \neq v_i$, then v_k dominates v_i.

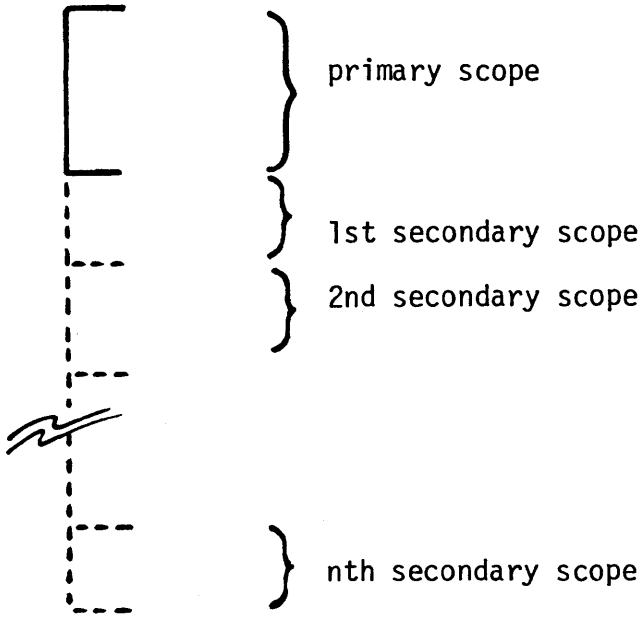
One of the important properties of the dominance relationship is that every node except the begin node (which has no dominators), has a unique direct dominator.

In the example shown in Figure 2, the subtrees rooted at 6 and 9 are located within the scope of 1 because vertices 6 and 9 have 1 as their common direct dominator. Similarly, the subtree rooted at 13 is located within the scope of 9 because 9 is the direct dominator of 11 whereas 1 is only a dominator of 11.

By placing the definition of each subtree within the scope of the direct dominator of its root, we guarantee that each subtree is enclosed within the most restrictive scope possible. This rule provides much of the power obtainable from the MTR.

In Figure 2, we saw examples of square brackets drawn with either a solid or broken line and defining two regions:

—a solid region containing the subtree attached to the root v_i and obtained by only traversing vertices with indegree less than two (i.e., with at most one predecessor). Notice that the vertices enclosed in angle brackets ⟨ ⟩ seem to violate that rule but these are only visited, not traversed. We call the solid region associated with v_i the primary scope of v_i.
—a broken region containing the subgraphs whose roots all have v_i as their direct dominator and indegree at least equal to 2. We call this broken region the secondary scope of v_i. In general, there may be not just one but several secondary scopes of a given vertex. Two subgraphs are in distinct secondary scopes of v_i if they are disconnected within the scope of v_i.



Within one of the secondary scopes of a given vertex, we expect to have p bracketed subgraphs, $p \geq 0$. Let us designate by $s_1, s_2, \ldots, s_p$ the roots of these p subgraphs and let $L(s_i)$ be the indentation level of s_i. By construction, we require that:

$$L(s_i) > L(s_j) \quad \text{if } R(s_i,s_j)=1 \wedge R(s_j,s_i)=0$$
$$L(s_i) = L(s_j) \quad \text{if } R(s_i,s_j)=1 \wedge R(s_j,s_i)=1$$

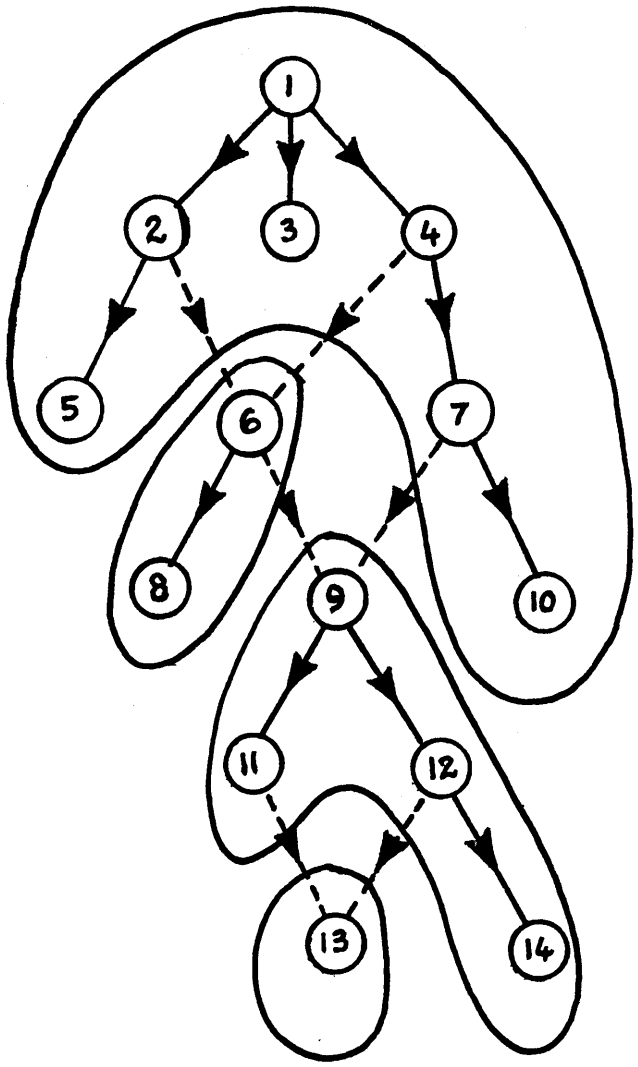
To make the solution practical, we also require that $\max_i(L(s_i))$ be minimum.

This indentation strategy has the advantage that for an acyclic subgraph, the root of each subtree is further indented than the root of any subtree that references it. Notice that if two roots s_i and s_j are at the same indentations level, then

$$(R(s_i,s_j)=0 \wedge R(s_j.s_i)=0) \vee (R(s_i,s_j)=1 \wedge R(s_j,s_i)=1)$$

i.e., either they call each other recursively, possibly via some indirect paths, or neither calls the other. As we mentioned earlier, most invocation digraphs of interest are acyclic or almost acyclic so that we expect most subgraphs that end up at the same indentation level not to reference each other.

Whether or not there are any recursive invocations can be quickly determined by scanning the definition/reference lists in the right hand margin of an MTR. Whenever a vertex with indegree greater than 1 is referenced (i.e., vertices in ⟨ ⟩), we append the MTR line number at which its definition (its subtree of descendants), is located. Conversely, whenever a vertex with indegree greater than 1 is defined, we append the list of MTR lines at which it is referenced. If a vertex with indegree greater than 1 happens to be on a cycle,




```
                +-----
1     1 ->      | 1
2     2         |     2
3     3         |           5
4               |           <6>
5     2         |     3
6     2         |     4
7               |           <6>
8     3         |           7
9               |                 <9>
10    4         |                 10
                |
                +-----
                .
                .   +-----
11    3         .   | 6
12    4         .   |     8
13              .   |     <9>
                .   +-----
                .
                .
                .       +-----
14    4         .       | 9
15    5         .       |     11
16              .       |           <13>
17    5         .       |     12
18              .       |           <13>
19    6         .       |           14
                .       |
                .       +-----
                .       .
                .       .   +-----
20    6         .       .   | 13
                .       .   +-----
                .       .
                .       +.....
                .
                +.....
```

Figure 2—MTR of an acyclic graph

a '*' precedes any MTR line number at which it is either defined or referenced.

Figure 3 shows an example of MTR containing recursive invocations. The reference lists contain '*' to indicate recursive paths and '|' to separate references located within the primary scope from those in the secondary scope. For instance:

REF AT (19| 72,228)

primary scope references	secondary scope references

Finally, each reference line can be supplemented by a left arrow showing which secondary scope contains the corresponding definition. An example is shown on Figure 4.

CONCLUSION

We have introduced the idea of Modular Tree Representation (MTR) of invocation digraphs. The algorithms necessary to build MTR's in practice are presented in Appendix A. The MTR provides a canonical representation for invocation digraphs.

```
                .   |   .   +-----
269    5        .   |   .   | FMT_SPECIFICATION  /P 60/                                   REF AT (268 | *270)
270             .   |   .   |      <FMT_SPECIFICATION  /P 60/ >                            DEF AT *269
271    6        .   |   .   |      REPEAT_SPEC  /P 61/
272             .   |   .   |          <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >           DEF AT 383
273    6        .   |   .   |      W  /P 62/
274             .   |   .   |          <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >           DEF AT 383
275    6        .   |   .   |      E  /P 63/
276             .   |   .   |          <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >           DEF AT 383
277    6        .   |   .   |      N  /P 64/
278             .   |   .   |          <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >           DEF AT 383
279    6        .   |   .   |      C  /P 65/
280             .   |   .   |          <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >           DEF AT 383
281    6        .   |   .   |      D  /P 66/
282             .   |   .   |          <UNSIGNED_INT_CONSTANT  /P 104/ >                   DEF AT 391
283    6        .   |   .   |      M  /P 67/
284             .   |   .   |          <UNSIGNED_INT_CONSTANT  /P 104/ >                   DEF AT 391
285    6        .   |   .   |      B  /P 68/
286    7        .   |   .   |          NONZERO_INTEGER_CONSTANT  /P 107/
287             .   |   .   |              <SIGN  /P 117/ >                                DEF AT 404
288             .   |   .   |              <DIGIT  /P 118/ >                               DEF AT 406
289    6        .   |   .   |      K  /P 69/
290             .   |   .   |          <INTEGER_CONSTANT  /P 106/ >                        DEF AT 393
291    6        .   |   .   |      H  /P 70/
292    7        .   |   .   |          PROCESSOR_CHARACTER  /P 114/
293             .   |   .   |              <APOSTROPHE  /P 115/ >                          DEF AT 396
294             .   |   .   |              <NONAPOSTROPHE_CHARACTER  /P 116/ >             DEF AT 397
                .   |   .   +-----
                .   |   .
                .   |   +.....
                .   |
                .   +-----
                .
                .
                .   +-----
295    3        .   | STATEMENT_FUNCTION_STATEMENT  /P 71/                                REF AT (13,28,43,54 |)
296             .   |      <EXPRESSION  /P 75/ >                                           DEF AT 307
297             .   |      <VARIABLE_NAME  /P 93/ >                                        DEF AT 362
298             .   |      <FUNCTION_NAME  /P 100/ >                                       DEF AT 368
                .   +-----
                .
                .
                .       +-----
299    5        .       | LEN_SPECIFICATION  /P 22/                                       REF AT (19 | 72,228)
300             .       |      <INT_CONSTANT_EXPR  /P 81/ >                                DEF AT 358
301             .       |      <NONZERO_UNSIGNED_INT_CONSTANT  /P 105/ >                   DEF AT 383
                .       +-----
                .
                .
                .       +-----
302    8        .       | FUNCTION_REFERENCE  /P 74/                                      REF AT (|199,*312,*319,
                .       |                                                                         *326,*334  )
303             .       |      <EXPRESSION  /P 75/ >                                       DEF AT 307
304             .       |      <ARRAY_NAME  /P 94/ >                                       DEF AT 364
305             .       |      <PROCEDURE_NAME  /P 98/ >                                   DEF AT 366
306             .       |      <FUNCTION_NAME  /P 100/ >                                   DEF AT 368
                .       +-----
                .
                .
                .       +-----
307    8        .       | EXPRESSION  /P 75/                                              REF AT (197,177,196,296,
                .       |                                                                         303)
308             .       |      <ARITHMETIC_EXPRESSION  /P 77/ >                            DEF AT 311
309             .       |      <CHARACTER_EXPRESSION  /P 84/ >                             DEF AT 325
310             .       |      <LOGICAL_EXPRESSION  /P 86/ >                               DEF AT 333
                .       +-----
                .
                .
                .       +-----
311    8        .       | ARITHMETIC_EXPRESSION  /P 77/                                   REF AT (|200,308,*313,
                .       |                                                                         *320,*342  )
312             .       |      <FUNCTION_REFERENCE  /P 74/ >                               DEF AT *302
313             .       |      <ARITHMETIC_EXPRESSION  /P 77/ >                            DEF AT *311
314             .       |      <ARRAY_ELEMENT_NAME  /P 90/ >                               DEF AT 345
315             .       |      <CONSTANT_NAME  /P 92/ >                                    DEF AT 389
316             .       |      <VARIABLE_NAME  /P 93/ >                                    DEF AT 362
317             .       |      <UNSIGNED_ARITHMETIC_CONSTANT  /P 103/ >                    DEF AT 370
                .       +-----
                .
                .
                .       +-----
318    8        .       | INTEGER_EXPR  /P 78/                                            REF AT (|106,144,183,
                .       |                                                                         187,206,*346,*349
319             .       |      <FUNCTION_REFERENCE  /P 74/ >                               DEF AT *302
320             .       |      <ARITHMETIC_EXPRESSION  /P 77/ >                            DEF AT *311
```

Figure 3—Partial MTR of the FORTRAN 77 syntax

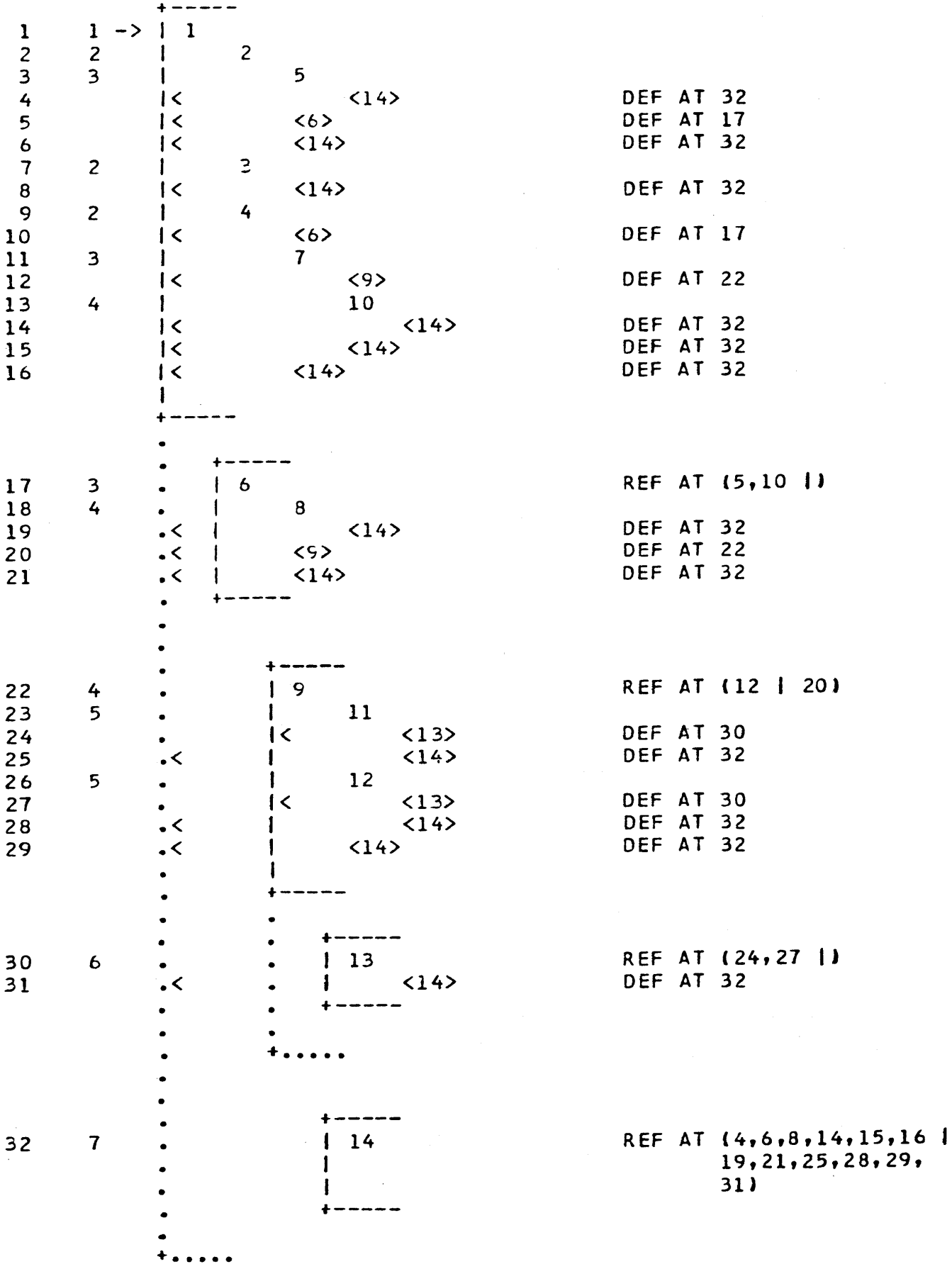


Figure 4—MTR with arrows

In order to experiment with the MTR concept on real projects we have built an omnibus MTR processor. With appropriate input/output interfaces, it can be used to process invocation graphs generated from PDL designs, HOL source code, load modules. . . . We have already used the processor in the following contexts:

—as an extension of the PDL[2] processor to help understand the morphology of a software design. The invocation graph is, in that case, the graph of segment references in a PDL design.
—as an aid in packaging software, particularly when grouping low level design segments into modules. The MTR provides an intrinsic decomposition into modules from which the actual module structure can be selected.
—as an analyzer of existing packages to help understand the hierarchy of calls to procedures, references to external structures. . . . An executable load module can be processed automatically to yield the MTR of all external references.
—as an aid in designing overlay structures by applying the MTR to the nonoverlay load module.

Based on our experiments so far, we can make the following observations:

- MTR's should be used during software development, particularly during the design phase. We have found that even a cursory examination of the computer generated MTR of a design could trigger design changes aimed at making the resulting MTR more "structured" i.e., more simply nested. For instance, realizing that a "GET NEXT CHARACTER" design segment was not nested within "GET TOKEN" simply because of an early initializing reference, suggested making "GET NEXT CHARACTER" a self-initializing module.
- MTR's can be effectively used during maintenance to help assess the impact of a change on an existing package. In that case, the MTR processor is applied to the executable load module directly.
- Practical MTR's can only be computer generated. This is due to the size of practical applications (the load module of the MTR processor itself has 90 vertices and almost 600 edges) and also to the high sensitivity of the MTR to small changes in the input graph.
- MTR's should only be computer maintained. This can be easily achieved because MTR's can be produced cheaply (for instance, the MTR of the MTR processor only takes 8.85 sec of computer time on an IBM 370/158 and a few thousand lines of print). Therefore, a new MTR can be produced every time a design change takes place or a new load module is generated.

There are several questions about the MTR which have not been answered yet. The first important question is to determine user acceptance of such a tool. Admittedly, an MTR is not as intuitively obvious as a graphical structure chart. It takes a little practice to learn how to read an MTR, i.e., to visualize the graph through the MTR and be able to use the MTR to reason about the graph. The second question is to investigate modifications and/or extensions which might make the MTR even more useful in practical situations. These may involve modifications of the display layout or extensions of the MTR capabilities. For instance, one might try visualizing the implicit scope of data items by indicating the direct dominator which controls all references to a given data item. Then, there is the fascinating possibility of applying the MTR to nonconventional areas. For instance, Figure 3 showed the partial MTR of the FORTRAN 77 syntax.

Clearly, further experimentation is needed to answer those questions and assess the full potential of the MTR technique.

BIBLIOGRAPHY

1. Aho, Alfred V. and Jeffrey D. Ullman, *The Theory of Parsing, Translation, and Compiling*, (Volume II: Compiling)
2. Caine, S. H. and E. K. Gordon, *PDL, A Tool for Software Design*, National Computer Conference, 1975, pp. 271, 276.
3. Dahl, O. J. and C. A. R. Hoare, "Hierarchical Program Structures," *Structured Programming*, Academic Press, New York, 1972, pp. 175-220.
4. DeRemer, Frank and Hans Kron, "Programming-in-the-Large versus Programming-in-the-Small," *ICRS Proceedings*, April 1975, pp. 114-121.
5. Haney, Frederick M., "Module Connection Analysis—A Tool for Schedule Software Debugging Activities," *AFIPS Conference Proceedings*, Vol. 41, 1972, pp. 173-179.
6. Haralick, R. M., "The Diclique Representation and Decomposition of Binary Relations," *JACM*, Vol. 21, No. 3, July 74, pp. 356-366.
7. Jackson, Michael, *Principles of Program Design*, Academic Press, 1975.
8. Jones, Douglas F., "A Programming Aid for Structured Programmers," *Proceedings of the ACM*, Nov. 1974, pp. 676-681.
9. Knuth, D., *The Art of Computer Programming*, Vol. 3 Addison Wesley, 1973.
10. Morven, Gentleman W. and Munro J. Ian, "Designing Overlay Structures," *Software Practice and Experience*, Vol. 7, 1977, pp. 493-500.
11. Parnas, D. L., "On the Criteria to be used in Decomposing Systems Into Modules," *COMM. ACM*, Vol. 15, No. 12, Dec. 1972, pp. 1053-1058.
12. Ramamoorthy, C. V., "Analysis of Graphs by Connectivity Considerations," *JACM*, Vol. 13, No. 2, April 1966, pp. 211-222.
13. Schneiderman, Ben, "A Review of Design Techniques for Programs and Data," *Software Practice and Experience*, Vol. 6, 1976, pp. 555-567.
14. Stevens, W. P., G. H. Myers, and L. L. Constantine, *Structured Design," IBM Systems Journal*, Vol. 2, 1974, pp. 115-139.
15. White, John R. and Richard K. Anderson, "Supporting the Structured Development of Complex PL/1 Software Systems," *Software Practice and Experience*, Vol. 7, 1977, pp. 279-293.

APPENDIX

A1 Construction of the MTR representation

This section deals with the algorithms required for generating MTR's. The construction of the MTR of an invocation graph involves primarily:

- finding the direct dominator of every vertex in G;
- performing an extended topological sorting of a digraph, acyclic or not;
- selecting entry points to force the invocation digraph to become weakly connected.

These algorithms are either classical or extension of classical ones.

It is important to notice that there exist several alternative formulations for most of the algorithms described here, depending upon the choice of data structure representation. For the MTR, we have chosen to use bit vectors and bit matrices rather than linked lists for representing the graphs and their ancillary data structures. Consequently, these algorithms are oriented towards parallel bit vector operations. For instance, for inding the descendants of a particular graph node, we rely on a bit vector "fusion" algorithm instead of the more familiar, stack oriented, depth first search algorithm.

A2 Computation of Direct Dominators

The algorithm for the computation of direct dominators can be found in Reference 1, for instance. Using PDL,[2] the algorithm can be informally written as follows:

```
• COMPUTE DIRECT DOMINATORS (DIRECT_DOM( ))
  LET THE VERTICES BE NUMBERED 1,2,. . . ,N WITH 1 THE BEGIN NODE
  INITIALIZE DIRECT_DOM (I)=1, I=1,2,. . . ,N
  DO FOR EACH VERTEX I EXCEPT 1
  :  DELETE TEMPORARILY VERTEX (I)
  :  BUILD LIST OF DESCENDANTS (D( )) REACHABLE FROM VERTEX (1)
  :  DO FOR EVERY VERTEX J IN D( )
  :  :  IF DIRECT_DOM(I)=DIRECT_DOM(J)
  :  :  :  SET DIRECT_DOM(J)=I
  :  :  ENDIF
  :  ENDDO FOR
  :  RESTORE VERTEX (I)
  ENDDO FOR
```

A3 Extended Topological Sorting

We now examine the algorithms required to determine the indentation level of a group of p shared subtrees having the same direct dominator. A directed graph G' with vertices $\{v'_1, v'_2, \ldots, v'_p\}$ is built as an invocation digraph where the p components would be the p shared subtrees. In other words, there is an edge between v'_i and v'_j if and only if the root of the jth subtree is referenced from within the ith subtree. Notice that G' is not in general weakly connected, but this is not required by the algorithm that follows.

The problem is to partition the set of vertices $\{v'_1, v'_2, \ldots, v'_p\}$ into the smallest number of disjoint subsets $s_1, s_2, \ldots, s_k$ such that

(a) if there is a path from v_i to v_j but not from v_j to v_i then

$$v'_i \epsilon s_\nu,\ v'_j \epsilon s_\mu \qquad 1 \leq \nu < \mu \leq k$$

(b) if there is a path both from v_i to v_j and v_j to v_i then

$$v'_i,\ v'_j \epsilon s_\nu \qquad 1 \leq \nu \leq k$$

If $v'_i \epsilon s_\nu$, the rank of the ith vertex is said to be ν. In the acyclic case, this problem is a simple topological sorting and we know[9] that condition (a) above is satisfied after the sort is performed. But if G' contains cycles, the classical topological sorting algorithm cannot be applied. We shall, therefore, extend it to handle cyclic graphs as well.

In the simple topological sort of an acyclic graph, vertices and their outgoing edges are deleted immediately after they have been processed, and one of the remaining vertices with zero indegree is selected as the next vertex to be processed. It is equivalent but computationally simpler to replace the deletion operation by the assignment to each vertex of a rank, initially 0. Thus, the search for a vertex with 0 indegree becomes the search for a vertex whose predecessors all have a non-zero rank.

```
• TOPOLOGICAL SORTING (ACYCLIC GRAPH)
  LET THE VERTICES BE NUMBERED 1,2,. . . ,N
  INITIALLY SET RANK(I)=∞, I=1,2,. . . ,N
  DO WHILE THERE ARE VERTICES WHOSE RANK( )=∞ STILL
  :  DO FOR EACH VERTEX I
  :  :  IF RANK(I)=0 THEN
  :  :  :  COMPUTE PRED( ), THE SET OF IMMEDIATE PREDECESSORS OF I, EXCLUDING I
  :  :  :  SET MXRNK=MAX(RANK(PRED(1)),RANK(PRED(2)), . . .)
  :  :  :  IF RANK≠∞ THEN
  :  :  :  :  RANK(I)=MX RNK+1
  :  :  :  ENDIF
  :  :  ENDIF
  :  ENDDO FOR
  ENDDO WHILE
```

To handle graphs with cycles, we modify the search strategy so that if the search for a vertex whose predecessors all have a non-zero rank fails, we then search for a maximal circuit whose predecessors, excluding nodes on the circuit, all have a non-zero rank. A circuit is maximal if it is not possible to find another circuit that includes at least one of its vertices as well as some other vertex that it does not already contain.
The algorithm can be carried out in two steps:

—replace each maximal circuit by a single substitute vertex having the same immediate predecessor and immediate successor vertices, thus obtaining an acyclic graph but with self loops;
—apply classical topological sorting to the derived acyclic graph, reflecting the rank assignment to every vertex of a maximal circuit when its substitute vertex is assigned a rank.

This approach provides both a proof and a computational method.

In order to find the maximal circuits we can use the following method:

We start with the simple adjacency matrix $A=(a_{ij})$ defined

```
                                                                        524,537,549,563,
              I    .                         I                          574,584,595,607)
              I    .                         I
              I    .                         +-----
              I    .
              I    +.....
              I
617    2 ->   I     ERRMON(IHCERRM)    /P 23/
618    2 ->   I     ERRTRA(IHCETRCH)    /P 24/
619    2 ->   I     FDIOCS#(IHCECOMH)    /P 32/
620      ->   I     <IHCCOMH2   /P 42/ >                               DEF AT 641
621    2 ->   I     IHCEFIOS   /P 44/
622           I         <IBCOM#(IHCECOMH)   /P 41/ >                   DEF AT 669
623           I         <IHCERRM   /P 47/ >                            DEF AT 662
624    3      I         IHCFIOS2   /P 50/
625    3      I         IHCUATBL   /P 52/
626    2 ->   I     IHCEFNTH   /P 45/
627           I         <ADCON#(IHCFCVTH)   /P 3/ >                    DEF AT 660
628           I         <FIOCS#(IHCEFIOS)   /P 34/ >                   DEF AT 661
629           I         <IBCOM#(IHCECOMH)   /P 41/ >                   DEF AT 669
630           I         <IHCERRM   /P 47/ >                            DEF AT 662
631           I         <IHCUOPT   /P 53/ >                            DEF AT 667
632    3      I         INTSWTCH(IHCECOMH)   /P 55/
633    3      I         INT6SWCH(IHCFCVTH)   /P 56/
634    2 ->   I     IHCETRCH   /P 48/
635           I         <ADCON#(IHCFCVTH)   /P 3/ >                    DEF AT 660
636           I         <FIOCSBEP(IHCEFIOS)   /P 35/ >                 DEF AT 668
637           I         <IBCOM#(IHCECOMH)   /P 41/ >                   DEF AT 669
638    2 ->   I     IHCFCVTH   /P 49/
639           I         <IBCOM#(IHCECOMH)   /P 41/ >                   DEF AT 669
640           I         <IHCERRM   /P 47/ >                            DEF AT 662
              I
              +-----
              .
              .    +-----
641    2 ->   .    I IHCCOMH2   /P 42/                                 REF AT (620 I *654)
642           .    I    <IBCOM#(IHCECOMH)   /P 41/ >                   DEF AT 669
643    3      .    I    IHCECOMH   /P 43/
644           .    I        <ADCON#(IHCFCVTH)   /P 3/ >                DEF AT 660
645    4      .    I        ADJSWTCH(IHCEFNTH)   /P 5/
646    4      .    I        ARITH#(IHCEFNTH)   /P 6/
647    4      .    I        FCVAOUTP(IHCFCVTH)   /P 26/
648    4      .    I        FCVCOUTP(IHCFCVTH)   /P 27/
649    4      .    I        FCVEOUTP(IHCFCVTH)   /P 28/
650    4      .    I        FCVIOUTP(IHCFCVTH)   /P 29/
651    4      .    I        FCVLOUTP(IHCFCVTH)   /P 30/
```

Figure 5—Partial MTR of an IBM 370 load module

by

$a_{ij}=1$ if there exists one or more directed edges from vertex v'_i to vertex v'_j, $i \neq j$
$a_{ij}=0$ otherwise

and proceed iteratively to obtain the reachability matrix R.

Let $a_{1k_1}a_{1k_2} \ldots a_{1k_m}$ be the only m non-zero values of the first row of A. We set

$$a_{1j}=a_{k_1j} \vee a_{k_2j} \vee \ldots \vee a_{k_mj} \qquad j=1,2,\ldots,n$$

If the new first row of A differs from the old one, we perform the operation again, using the $m'-m$ new non-zero entries in the first row. If the new first row of A is identical to the previous one, the first row of R has been obtained. We now proceed to obtain the second row, then the third. . . . This process is often known as fusion.

When all n rows have been computed, the non-zero diagonal entries of R indicate those vertices that belong to some directed circuits. If we now compute

$$R=R \wedge R^T$$

where the logical product is taken element by element, the resulting R matrix has the following structure:

if $r_{ii}=1$ vertex i is on a directed circuit
if $r_{ij}=1$ then $r_{ji}=1$ by construction, v'_i and v'_j are on the same maximal circuit.

A4 Selecting entry points

The method for finding descendants that we have just described can now be used to solve a problem deferred earlier, namely, the appropriate selection of vertices to make the graph weakly connected. Such a selection is necessary when the initial graph still has more than one component after all vertices with indegree 0 have been linked to the artificially created begin node v_0. Let us assume that there are p such remaining unreachable vertices $V=\{v_{i_1}, v_{i_2}, \ldots, v_{i_p}\}$. The problem is to find the minimal number k of vertices from V sufficient to cover all vertices in V; i.e., all vertices in V are reachable from 1 or more of the k selected "entry" vertices. Notice that the solution is, in general, not unique since any vertex on a circuit may be replaced by any other on the same circuit.

The following algorithm finds the entry vertices:

```
LET E( ), R( ), C( ) BE P ELEMENT VECTORS WITH VALUE 0, 1
INITIALLY SET E( )=0, C( )=0
DO WHILE THERE EXISTS I SUCH THAT C(I)=0
    FIND THE DESCENDANTS (R( )) OF VERTEX (I)
        I.E. R(J)=1 IF VERTEX J IS REACHABLE FROM I
    SET E( )=(E( )∧7R( ))
    SET E(I)=1
    SET C( )=C( )∨E( )∨R( )
ENDDO WHILE
. . . THE ENTRY VERTICES ARE I SUCH THAT E(I)=1, I=1, . . . , p
```

The selection of entry vertices is illustrated on Figure 5. The arrows '→' in the left margin indicate addresses of the IBM OS/VS FORTRAN run time library which are not directly referenced within the load module.

The impact of program and programmer characteristics on program size

by EARL CHRYSLER
University of Wisconsin-Parkside
Kenosha, Wisconsin

INTRODUCTION

Since the first efforts of Farr and Zagorski[8] to those of Nelson[14] with the System Development Corporation and the more current, on-going work of Walston and Felix of IBM,[22] there has been the desire to develop a method of predicting the ultimate size of a program before the programming effort begins.

A technique for predicting the number of source language instructions necessary to meet a set of program specifications could be of value to decision-makers in facilities which allow programmers to store source language versions of programs on disk for online debugging and revision. Management could then estimate the amount of disk storage area necessary to support the development of each new program.

Management personnel would also find it valuable to be able to predict the amount of object code a specific set of program specifications will require for differing reasons. The basic question of whether the program will fit in main memory would be answerable for those operating in a constrained environment. For those utilizing partitioned memory, the impact upon the workload in each partition would be predictable. For facilities operating in a virtual storage type environment, the ability to predict the number of pages a program will require would be of significant value in determining if current disk storage capacity would be adequate to support projected programs.

THE PROGRAMMING ENVIRONMENT

The model of the programming environment used in this study is the one successfully utilized by this researcher to investigate the area of programmer productivity, shown in Figure 1. The rationale for this approach is straightforward. A set of program and programmer characteristics have previously been analyzed regarding their impact on program development time. Most of the program processing characteristics and all of the programmer experience characteristics hypothesized as being important were found to have a significant effect on program development time.[4] If one accepts the premise that longer program development time is related to larger programs, then it seems reasonable to assume that the variables found related to program development time should, more likely than not, also be related to program size.

PROGRAM AND PROGRAMMER CHARACTERISTICS

The purpose of this research was to determine the effect of program and programmer characteristics on program size, both in terms of source language lines of code and object language memory storage measured in bytes. As a consequence, all the remaining sources of variance in the programming environment had to be held constant. The research setting used was one where programmers coded offline and only COBOL programs were considered. Since this is the most typical business programming setting, it was believed the findings of the research would then have more general applicability. Since the study was conducted in a specific data processing facility within one firm, it was felt that the hardware and organizational variables were then adequately controlled for. Only the programming problem and the programmer, therefore, remained as sources of variance in program size.

In order to develop a reliable technique for predicting program size, the following questions must be answered:

1. What characteristics of the programming task, which can be objectively measured before the task is begun, are significantly correlated with program size?
2. What characteristics of the programmer, which can be objectively measured before the task is begun, are significantly correlated with program size?

When considering the characteristics of processing requirements of a program which one could reasonably expect to have an impact on the size of the resulting program, the same variables examined for possible impact on program development time in previous research[4] were believed worthy of analysis. Those program characteristics are:

1. Total number of input files
2. Total number of types of input records
3. Total number of unique fields of input records

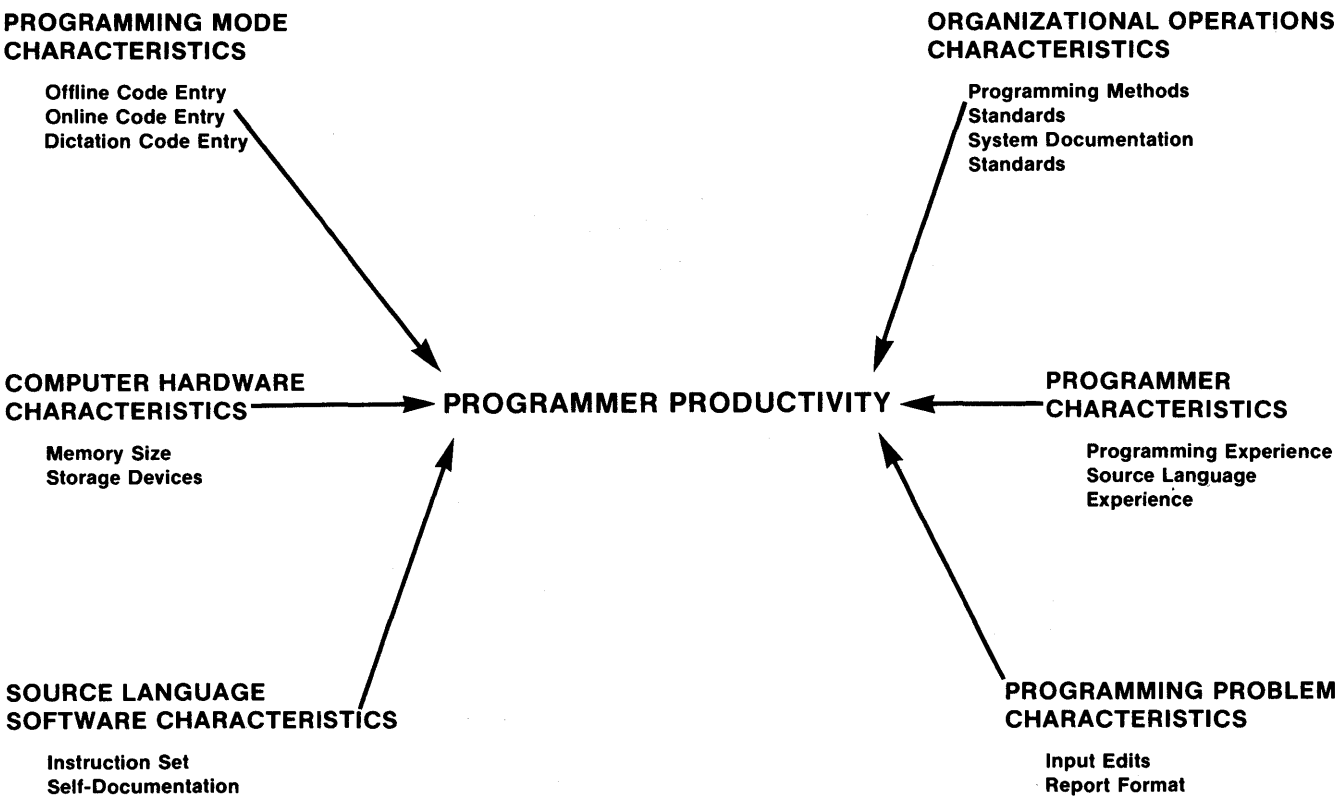


Figure 1—The computer programming environment

4. Total number of edits to be performed on input data
5. Total number of output files and report formats
6. Total number of types of output records
7. Total number of unique fields of output records
8. Total number of output fields not appearing as fields on input records
9. Total number of input files for which presence is optional
10. Total number of input records for which presence is optional
11. Total number of output files for which development is optional
12. Total number of output records for which development is optional
13. Total number of control breaks and/or totals specified for all output reports
14. Optional control breaks and/or totals
15. Mathematical operations other than developing totals

The programmer characteristics considered as possibly having an impact on program development time were similarly believed to be those with the greatest likelihood of having an impact on final program size. Those programmer variables are:

1. Number of months of programming experience
2. Number of months of programming experience using the COBOL language
3. Number of months of experience using the specific COBOL language compiler to be used for the subject program
4. Number of months experience programming business applications
5. Number of months experience programming for the current employer

METHODOLOGY

The purpose of this research was to examine the relationship between various processing characteristics of programs and experience characteristics of programmers and program size. The ultimate objective was to develop a viable technique for predicting the amount of source language lines and object language bytes a program would eventually become.

Research hypotheses

Since there is little research which verified that the program and programmer characteristics discussed have an effect on program size, this research is actually testing the null hypothesis, for each specified program variable, that the variable has no effect on program size versus the alternative hypothesis that the variable is associated with an increase in program size, that is, there is a positive correlation between each program variable and program size.

The specified programmer variables, it is hypothesized, will also be found to be related to program size, but the coefficients will have a negative sign, that is, there will be a negative correlation. Increasing values for programmer variables will be associated with reductions in program size.

Data collection techniques

The first step was to locate a data processing facility which employed a group of computer personnel to create new COBOL programs in an offline environment. The subject firm was selected for two additional reasons. First, there was a wide range of experience among the members of the programming staff, from trainees with eight months experience to senior programmers with over ten years of experience. Second, the firm also performed contract programming for other firms. This implied that one could expect more variation among the types of programs developed at this facility than in an installation where programs were created only for internal systems.

It should be restated here that the program variables to be measured must, if a predictive technique is to be reliable and effective, be measurable *before* programming begins. Since the quality of pre-programming documentation at the subject firm was not adequate to allow such measurements to be made, it was necessary for the researcher to examine the COBOL source statement or compilation listing for each program and measure the variables of interest from the listing. While this was necessarily tedious, it also improved the control over the research data. This improvement in control was accomplished because one could verify that the program was indeed coded in COBOL. Also, one was able to detect and remove from the analysis any program which utilized the Report Writer feature and was therefore not comparable to the other programs. In order to measure the value of the programmer variables, a questionnaire which asked the programmers to report their programming experience was circulated to all programmers by the programming manager.

The size of the program in source language lines of code was measured from the listings, subdivided into Data Division lines and Procedure Division lines. The number of bytes of storage required to store the object code was also taken from the compilation listing and was similarly subdivided into Working Storage bytes and Procedure Division bytes.

Statistical analyses to be performed

The first step would be to provide the data for program size and program variables to a stepwise multiple linear regression computer program for analysis of the interrelationships present. The identical procedure would be followed relating the size of a program to each of the programmer variables for the programmer who created the program. With regard to the programmer variables, however, it would be necessary to adjust the values depending upon the date the programmer started the programming assignment. That is, the value of each programmer experience variable had to be reduced to what the value would have been when the programmer started the task since the variables are all length-of-time experience values.

The last step would be to relate program size to the group of program variables and programmer variables simultaneously to develop a specific predictive equation for use as an estimating device.

Limitations of the study

The major limitation of the study is the lack of enforcement of programming standards at the firm used in the study. Each programmer had his own style in terms of the appearance of the source language code. For example, some programmers would place both the PICTURE and VALUE clauses for a data field on one card or line of code while others would place the VALUE clause entry on a separate card. Another observable difference was that some programmers would place only one verb per line of code in the Procedure Division while others would place as much code as possible on each card. Another limitation is the accuracy with which programmers reported their experience on their questionnaire. Aside from the fact that inaccuracies may be intentionally introduced into the data, there is the problem that a subject was asked to recall how long he performed a given type of programming. Further, if a subject performed some functions other than those of a programmer during a given time period, he was also asked to parcel out that portion associated with only the programming effort. Since future applications of this technique will be using input with these same shortcomings, however, perhaps the reliability of the technique will be improved by the high similarity between the experimental environment and the application environment.

Some assumptions which could be classified as limitations are that the subject programmers were provided with comparable documentation of the processing requirements for each program and that the programs developed were of comparable quality.

It should be noted that the items listed as limitations are sources of the error or unexplained portion of the variance to be found in the analysis. Therefore, the mathematical statements of the relationships found between the sample variables could be considered to be underestimates of the true strength of the relationships between the population variables.

RESEARCH FINDINGS

A total of thirty-six programs were found to qualify for statistical analysis by being written in COBOL and containing no use of special compiler features which made them unusual in terms of coding required or logical construction. The results of the analysis are stated below.

TABLE I—Correlation Coefficients of Program Characteristics with Data Division Source Statements

Program Characteristic	r
Input Files	.19910
Input Records	.43704****
Input Fields	.57080*****
Input Edits	.19552
Output Files and Report Formats	.47542****
Output Records	.62756*****
Output Fields	.72589*****
Output Fields without corresponding input fields	.40557***
Optional Input Records	.31012*
Optional Output Files and Report Formats	−.15245
Optional Output Records	.19226
Control Breaks and Totals	.50122****
Optional Control Breaks and Totals	−.08216
Mathematical Operations	.37451**

*****significant at .0005 level
****significant at .005 level
***significant at .01 level
**significant at .025 level
*significant at .05 level

TABLE III—Correlation Coefficient of Each Program Characteristic with Working Storage Bytes

Program Characteristic	r
Input Files	.58296***
Input Records	.10407
Input Fields	−.03186
Input Edits	−.03899
Output Files and Report Formats	−.00679
Output Records	.07999
Output Fields	−.00548
Output Fields without corresponding input fields	−.04902
Optional Input Records	−.05390
Optional Output Files	.28230*
Optional Output Records	.15972
Control Breaks and Totals	−.07603
Optional Control Breaks and Totals	.34933**
Mathematical Operations	−.05159

***significant at .0005 level
**significant at .025 level
*significant at .05 level

Program characteristics

As can be seen in Table I, the most important program variable in terms of its impact on Data Division source language lines of code was output fields, closely followed by output records, input fields and control breaks and totals. All of these variables were significant at the .005 level. As can be seen, most of the program variables were related to an increase in the size of the source code found in the Data Division.

As shown by Table II, the three program variables which one could reasonably associate with computations and logical complexity emerged as having the greatest effect on the amount of source language code found in the Procedure Division. All three, mathematical operations, output fields without corresponding input fields and control breaks and totals exhibited correlation coefficients of over .9.

In terms of the impact of the program characteristics on the amount of object code generated, Tables III and IV provide considerable insight. The only variables found to be strongly related to an increase in the amount of Working Storage bytes needed for a program, in order of their strength, were input files, optional control breaks and totals and optional output files. With regard to the Procedure Division bytes required, the same variables, mathematical operations, output fields without corresponding input fields and control breaks and totals are the only program characteristics which appear to have an effect.

Programmer characteristics

Although thirty-six programs were available for analysis regarding the impact of program characteristics on programming time, five of those had to be removed from the analysis of programmer variables because the responsible programmers had left the firm and their characteristics were not

TABLE II—Correlation Coefficients of Program Characteristics with Procedure Division Source Statements

Program Characteristic	r
Input Files	.02436
Input Records	.09295
Input Fields	.01439
Input Edits	.02551
Output Files and Report Formats	−.03108
Output Records	.10602
Output Fields	.08491
Output Fields without corresponding input fields	.95237*
Optional Input Records	.04856
Optional Output Files and Report Formats	−.07483
Optional Output Records	.03558
Control Breaks and Totals	.93371*
Optional Control Breaks and Totals	−.04447
Mathematical Operations	.97183*

*significant at .0005 level

TABLE IV—Correlation Coefficient of Each Program Characteristic with Procedure Division Bytes

Program Characteristic	r
Input Files	.02861
Input Records	.04875
Input Fields	−.02495
Input Edits	.01090
Output Files and Report Formats	−.05898
Output Records	.07655
Output Fields	.04933
Output Fields without corresponding input fields	.96540*
Optional Input Records	.00096
Optional Output Files	.06858
Optional Output Records	.01659
Control Breaks and Totals	.94325*
Optional Control Breaks and Totals	−.03942
Mathematical Operations	.98177*

*significant at .0005 level

TABLE V—Correlation Coefficients of Programmer Characteristics with Data Division Source Statements

Programmer Characteristic	r
Months of experience programming at this facility	−.46063**
Months of experience programming business-oriented applications	−.44684**
Months of experience programming (total)	−.45020**
Months of experience programming with COBOL compilers	−.27983
Months of experience programming with COBOL compiler in use at this facility	−.34568*

**significant at .01 level
*significant at .05 level

TABLE VII—Correlation Coefficients of Programmer Characteristics with Working Storage Bytes

Programmer Characteristic	r
Months of experience programming at this facility	−.34113*
Months of experience programming business-oriented applications	−.35143*
Months of experience programming (total)	−.36714**
Months of experience programming with COBOL compilers	−.30781*
Months of experience programming with COBOL compiler in use at this facility	−.28129

**significant at .025 level
*significant at .05 level

measurable. For the thirty-one programs, the four measurements of program size were individually regressed against the programmer characteristics and the correlation coefficient between each characteristic and each component of program size was computed.

The findings shown in Table V indicate four of the five programmer experience variables are related to reductions in the size of the Data Division source code. The three strongest influences appear to be total programming experience, business programming experience and, curiously, experience at this facility. While one can easily accept the business programming and total programming experience variables as important, the relationship between experience at this facility and reduced Data Division code is less clear and may be unique to this data processing installation for reasons unknown at this time.

The analysis of programmer characteristics and source statements found in the Procedure Division provided the most unexpected results. As the correlation coefficients in Table VI indicate, there appears to be no significant relationship between any programmer experience characteristic and the size of the source language Procedure Division. Since previous research[4] has found these programmer variables are all significantly related to reduced program development time, this latter finding raises an interesting question. Does increased experience of the types listed give one the ability to develop the same amount of processing logic code as another less-experienced programmer in significantly less time? That is the implication of these findings.

When relating the programmer characteristics to the object code equivalents of the Data and Procedure Divisions source code, the findings were as shown in Tables VII and VIII. The findings with regard to Working Storage bytes differ only slightly with the source code findings. Total programming experience, business programming experience and programming experience at this facility, however, remain the most important variables. With regard to Procedure Division bytes, the findings are identical to those for the impact of the variables on source code, that is, no significant relationships were found.

TABLE VI—Correlation Coefficients of Programmer Characteristics with Procedure Division Source Statements

Programmer Characteristic	r
Months of experience programming at this facility	.04937
Months of experience programming business-oriented applications	.00333
Months of experience programming (total)	−.00152
Months of experience programming with COBOL compilers	.08817
Months of experience programming with COBOL compiler in use at this facility	.11644

TABLE VIII—Correlation Coefficients of Programmer Characteristics with Procedure Division Bytes

Programmer Characteristic	r
Months of experience programming at this facility	.07447
Months of experience programming business-oriented applications	.03104
Months of experience programming (total)	.02666
Months of experience programming with COBOL compilers	.11312
Months of experience programming with COBOL compiler in use at this facility	.14020

Estimating equations

In order to develop equations for estimating the two components of source program size and the two components of object program size, one would use a stepwise multiple linear regression analysis program. The logic of such a program is to continue to add each of the predictor variables into the estimating equation one by one, each time entering that remaining variable which most improves the accuracy of the estimate. A point is eventually reached where the additional value of the next variable in the equation is small and, more importantly, not of statistically significant value. The equations shown here are those which in each case were, statistically, optimum. In order to develop such equations, the variables to be subjected to analysis would be those which appeared to have a significant impact.

The approach decided upon was to first use only the group of program characteristics to develop equations for the source and object code of the Data and Procedure Divisions. The analyses would then be repeated allowing programmer characteristics to enter the equations to determine if predictive power was increased. The equation shown in Figure 2 for estimating Data Division source lines had a multiple

Variable Name	Regression Coefficient
Constant	−26.41
Output Fields	1.69
Output Fields without corresponding input fields	4.84
Input Files	35.45
Input Fields	.59
Mathematical Operations	−.62
Input Records	6.03

Figure 2—Equation for predicting data division source lines with program characteristics

correlation coefficient of .94 and, therefore, the variables included explain about 88 percent of the variance found in the number of source lines appearing in the Data Division of the sample programs. The optimum equation for the Procedure Division source lines, shown in Figure 3, using only program characteristics, had a multiple correlation coefficient of .997.

As an example of the use of these equations, assume one wished to estimate how many source lines of code would be required for a program with the following characteristics:

8 mathematical operations
20 output fields
1 input record type
0 optional output records
0 input edits
4 output fields without corresponding input fields
18 input fields
and
1 input file

By placing those values in their appropriate positions in the equations one obtains a value of about 74 lines of Data Division code and 140 lines of Procedure Division code. The program would then be estimated at about 214 lines of code plus the necessary Identification and Environment Division entries, which typically show little variance. Upon allowing the programmer characteristics to enter the equations it was found that, although the equations became more complex, no significant improvement could be made in the estimate.

The next analysis concerned the object code components of the program. The program variables which developed the best equation, statistically speaking, are shown in Figures 4 and 5. The multiple correlation coefficient for the Working Storage bytes equation was .66, accounting for only about 43 percent of the variance in this component of program size. Once again it was found that allowing programmer characteristics to enter the equation provided little additional predictive power.

Variable Name	Regression Coefficient
Constant	−6019.57
Input Files	7377.95
Optional Control Breaks and Totals	1802.77
Output Files and Report Formats	−1903.16

Figure 4—Equation for predicting working storage bytes with program characteristics

The optimum equation for predicting Procedure Division bytes using only program characteristics, shown in Figure 5, had a multiple correlation coefficient of .998. This implies that the number of bytes required in the Procedure Division is apparently predictable from knowledge of the program characteristics appearing in Figure 5. Programmer characteristics were once more found of no significant value to the equation. A necessary but realistic assumption is that the programming specifications at a data processing facility are of sufficient quality that the program characteristics in the equations are defined before programming is begun. Given this assumption is met, the equations presented in Figures 4 and 5 could be used to predict the amount of storage a program will require by placing the program characteristics in the equations and computing the final value as shown previously for the source statement portions of a program.

SUMMARY AND CONCLUSIONS

Summary

The purpose of this research was to analyze the relationships between characteristics of a programming problem and the responsible programmer and the amount of source and object code necessary to complete the programming problem. For each program characteristic it was hypothesized that

Variable Name	Regression Coefficient
Constant	−31.09
Mathematical Operations	2.97
Output Fields	1.81
Input Records	37.02
Optional Output Records	45.98
Input Edits	−2.32
Output Fields without corresponding input fields	2.90

Figure 3—Equation for predicting procedure division source lines with program characteristics

Variable Name	Regression Coefficient
Constant	−38.64
Mathematical Operations	50.37
Output Fields	26.32
Optional Output Records	626.73
Input Records	659.70
Output Fields without corresponding input fields	63.57
Input Edits	−19.68
Optional Input Records	−218.67

Figure 5—Equation for predicting procedure division bytes with program characteristics

an increase in the value of that variable would lead to an increase in each aspect of program size. For each programmer variable it was believed that an increase in the value of the variable would contribute to a decrease in the program size measurements.

Simple linear correlation coefficients were computed to determine the direction and magnitude of the relationships between the variables in the analysis. In order to develop predictive equations for the components of program size, stepwise multiple linear regression analyses were performed.

Conclusions

Of the fifteen program characteristics examined, nine were found to have correlation coefficients with Data Division source statements significant beyond the .05 level, while only three were found correlated with Procedure Division source statements. The strength of the latter relationships, however, was significant beyond the .0005 level.

In terms of their impact on Working Storage bytes of object code, only three program characteristics were found to be significant, varying in strength from only .05 to .0005 levels of significance. Three different program characteristics, however, were found to be related to Procedure Division bytes of object code, all at the .0005 level of significance.

While all five programmer characteristics were negatively correlated with both Data Division source statements and Working Storage bytes as hypothesized, only four were significant beyond the .05 level. None of the five programmer experience characteristics could be shown to cause a reduction in either the Procedure Division source lines of code or the Procedure Division bytes of object code, a finding worthy of note.

When the program and programmer variables were used to develop predictive equations for the four size components, strong multiple correlation coefficients were found without programmer characteristics present in the equations. The inclusion of programmer characteristics into the equations appeared to provide no significant increase in predictive power.

The findings of this study imply that one could, with a reasonable degree of accuracy, predict the two major source and object components of program size of a program before programming begins by developing equations of the type presented here with data collected at the facility where they are to be used for estimation. As new programs are developed, their program and programmer characteristics and four size components should be measured and added to a history data bank for the firm. These data should then be periodically used as input to a stepwise multiple linear regression analysis program. As the firm continues to increase the number of program data points in its data bank, it will be capable of predicting the size of new programs with increasing accuracy.

BIBLIOGRAPHY

1. Boehm, Barry W., "Software and Its Impact: A Quantitative Assessment," *Datamation,* May 1973, pp. 48-59.
2. Brandon, Dick H., *Management Standards for Data Processing,* New York, D. Van Nostrand Company, Inc., 1963.
3. Brooks, Frederick P. Jr., "The Mythical Man-Month," *Datamation,* December 1974, pp. 45-52.
4. Chrysler, Earl, "Some Basic Determinants of Computer Programming Productivity," accepted for publication by *Communications of the ACM.*
5. Daniel, Wayne W. and James C. Terrell, *Business Statistics,* Boston, Houghton Mifflin, 1975.
6. Day, Carolyn, "Why Document?" *Data Management,* January 1975, pp. 6-9.
7. Draper, N. R. and H. Smith, *Applied Regression Analysis,* New York, John Wiley and Sons, Inc., 1966.
8. Farr, Leonard and Henry J. Zagorski, "Factors that Affect the Cost of Computer Programming: A Quantitative Analysis," Santa Monica, California, System Development Corporation, August 1964, pp. 59-86.
9. Gayle, John B., "Multiple Regression Techniques for Estimating Computer Programming Costs," *Journal of Systems Management,* February 1971, pp. 13-16.
10. Howard, Marvin, "Organizing and Managing the Programming Effort," *Data Processing Digest,* November 1968, pp. 1-18.
11. Keider, Stephen P., "Why Projects Fail," *Datamation,* December 1974, pp. 53-55.
12. Krauss, Leonard I., *Administering and Controlling the Company Data Processing Function,* Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1969.
13. La Bolle, Victor, "Development of Equations for Estimating the Costs of Computer Program Production," Santa Monica, California, System Development Corporation, April 1966.
14. Nelson, E. A., "Management Handbook for the Estimation of Computer Programming Costs," Santa Monica, California, System Development Corporation, March 1967.
15. Nunnally, Jum C., *Psychometric Theory,* New York, McGraw-Hill, 1967.
16. Ridge, Warren J. and Leann E. Johnson, *Effective Management of Computer Software,* Homewood, Illinois, Dow Jones-Irwin, Inc., 1973.
17. Rothery, Brian, *Installing and Managing a Computer,* New York, Brandon Systems Press, 1969.
18. Sackman H., W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM,* January 1968, pp. 3-11.
19. Scott, Randall F. and Dick B. Simmons, "Programmer Productivity and the Delphi Technique," *Datamation,* May 1974, pp. 71-73.
20. Upton, Molly, "Software Firms Rate High on Pay Scale," *Computerworld,* January 1, 1975, pp. 24-25.
21. Weinwurm, George F. (ed.), *On the Management of Computer Programming,* Princeton, New Jersey, Auerbach Publishers, Inc., 1970.
22. Walston, C. E. and C. P. Felix, "A Method of Programming Measurement and Estimation," *IBM Systems Journal,* Jan-Mar, 1977, pp. 54-73.
23. Wolverton, Ray W., *The Cost of Developing Large Scale Software,* Operational Software Operations, TRW Systems Group, One Space Park, Redondo Beach, California, 1972.

Multiprocessing made easy

by RONALD J. PRICE
Perkin-Elmer Data Systems Group
Tinton Falls, New Jersey

INTRODUCTION

The concurrent execution of processes on multiple machines has appealed to system architects for quite some time.* In fact, it is not such a new concept as often thought. According to P. Enslow, "The first operational 'true' multiprocessor was delivered in 1962."[1]

Increased throughput is usually the major objective of multiprocessing, but there are other advantages over interleaved, multi-programming of a uniprocessor that oftentimes take on greater importance.

For example:

- Resource sharing
- Responsiveness to external events
- Expandability in a stepwise manner
- System modularity
- Survivability

There are a number of reasons why multiprocessing is not as popular as perhaps it could be, as Enslow aptly points out in Reference 2. One big problem is software. Program development is difficult because the behavior of concurrent programs is time-dependent and seldom reproducible with a given set of input data. Add parallelism with increased possibilities for time-dependent interactions and deadlock situations, and we have a most complex problem. The lack of software for the number of available multiprocessor systems[1] attests to this fact.

Help is on the scene, though!

A new design concept that combines shared data structures and the operations permitted on them in a single syntactical construct called a "monitor" has been suggested in the literature by C. A. R. Hoare[3] and Per Brinch Hansen[4] as an effective tool for building concurrent system programs. Brinch Hansen based his Concurrent Pascal language on this principle. He extended the sequential program properties of the Pascal language with concurrent programming tools called processes and monitors.[5] This paper is based on the constructs in Concurrent Pascal, although the research works of E. W. Dijkstra[6] and N. Wirth[7] are also relevant.

Evidence exists that the monitor concept, particularly as implemented in Concurrent Pascal, is a powerful tool. Brinch Hansen recorded improvement in programmer productivity while building a complete operating system with his language.[8] The utility of the language has also been tested for many diverse applications.[9]

From personal experience, the author has found that the mere act of decomposing a concurrent problem into logical processes and monitor modules, irrespective of how they might be implemented, is an excellent design methodology. Furthermore, once system facilities are provided to manage process and monitor programs, the systems programmer need no longer suffer through analyses of time-dependent race conditions and interprocess communication protocols. Performance need not be sacrificed either. In fact, the author discovered that performance improved, probably because of a better overall system design.

The monitor concept is a relatively new idea and some weaknesses can be identified. Several researchers seek additional language constructs and operators (for examples see References 9 through 14). This is to say nothing of the age-old question whether compilers can produce optimum time and space code, nor the difficulties of implementing language support for multiprocessing. On the other hand, the system programmer need not wait for the ultimate language. He can reap the benefits of this new software technology for current problems even if he has to use assembly language. All he has to do is build a kernel facility, which manages process and monitor programs, and adhere to the programming conventions promulgated by these experimental languages.

The next section presents a brief explanation of the monitor concept and how it relates to multiprocessing applications. This is followed by a section on implementation techniques for supporting multiprocessors. The last section illustrates an example application and its solution. By employing two assembly language solutions, one with monitors and one without, the example evaluates the fundamental utility of the monitor concept for solving concurrent/multiprocessing problems; i.e., biases of a compiler for generating code and for enforcing programming conventions are not involved. The example also illustrates the value of the monitor as a systems structured programming tool by solving the concurrent problem in stages.

* Tightly coupled, multi-port shared memory systems are assumed in this paper, although much of the presentation equally applies to other types of multi-processor systems.

MONITOR CONCEPT

The reader should consult the references for an in-depth explanation of monitors and how they are employed in concurrent programs. Briefly though, monitors consist of shared data structures combined with a well-defined set of procedures permitted to operate on the shared data. In this context, processes are sequential programs which synchronize their operations and get access to shared data structures by calling monitor procedures in a manner much like subroutines. A run-time kernel executive enforces mutual exclusion on access to a monitor from competing processes.

The simple example in Figure 1 illustrates how processes can asynchronously exchange messages through monitors. Input and output operations are overlapped concurrently with job processing. The INPUT process READs a message from the IN_TERMINAL and PUSHes it on a buffer stack for the JOB process in the IN_BUFF monitor; a similar operation takes place on output. The monitors can be represented by ordinary program code except they are passive modules and do not execute until called. They contain the procedures (for example, PUSH and GET) that operate on the shared buffer stacks. They also contain the counters and flags needed to control the operation.

Inter-process synchronization is achieved through the use of a *delay-queue* construct by which a monitor procedure can *delay* its calling process or *continue* another process waiting in a *delay-queue*. For example, in Figure 1 the JOB process knows nothing of the situation of the INPUT and OUTPUT processes. If it tries to GET a message when none is available, it is *delay*ed by IN_BUFF until *continue*d when one is supplied by the INPUT process. Likewise, if the JOB process tries to PUT an output message when no buffer is available for the data, it is *delay*ed by OUT_BUFF until the OUTPUT process FETCHes a previous message and frees a buffer.

Other applications for monitors include classical reader-writer problems, process synchronization on events, resource sharing, priority scheduling, and so on. The concept is valid because processes, containing their own private data structures, cannot directly intercommunicate, change control (shared) data, or access a shared resource; to do so they must call a monitor, and the kernel permits a monitor to service only one process at a time.

The technique of describing a system program in terms of monitors and sequential processes can be used to represent a multiprocessing program. Any process is a candidate to be executed in parallel with others on separate processors. The three processes in Figure 1 could be executed literally simultaneously on different processors with no more difficulty (functionally) than if interleaved concurrently on one processor. Their combined execution becomes serial only when interacting through the same monitor.

In a multiprocessor configuration, monitor code can be executed out of shared memory or be replicated in each processor's private local memory. That is, if necessary, an instance of a given monitor can be represented on multiple machines. The separate representations can even be implemented differently (for example, microcoded) in different processors provided they execute the procedures as required by their respective processes. Monitor code can be duplicated in different processors because the run-time kernel permits only one version to execute at any given time. However, only one copy of a monitor's variable data structures can exist and it must be located in shared memory to be accessible by the different processors.

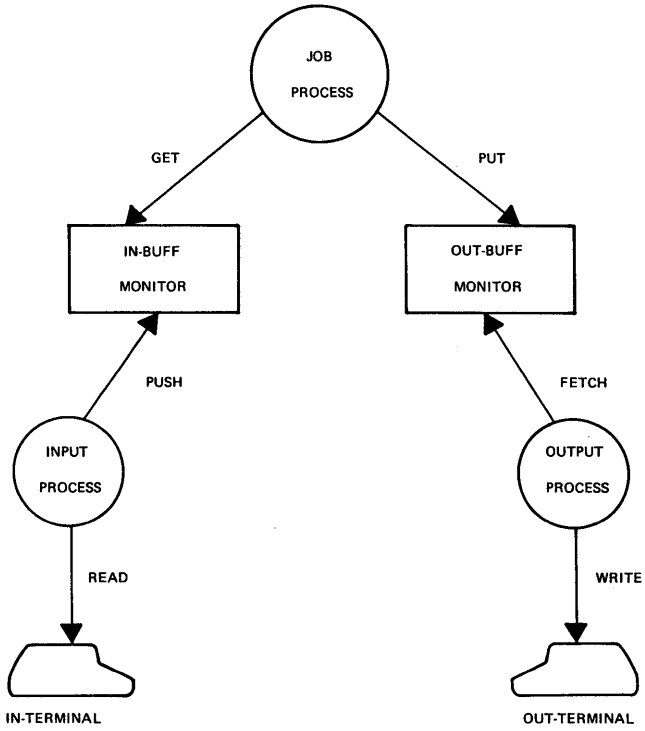


Figure 1—Example application of monitors

Each physical processor contains a run-time kernel. Collectively, these kernels create a virtual machine which supports an integral distributed processing system. In fact, control over processes for scheduling and dispatching is shared among the processors and consequently the virtual machine can be considered truly distributed.[15]

For example, the collection of the processes and monitors in Figure 1 represents a concurrent system program that accomplishes an application function (albeit rather trivial in this case). It consists of disjoint modules which run in harmony as one program where parts of it may execute in parallel. No single processor controls the execution of the others. With adequate kernel support, the application program can be executed on multiple machines or on a single machine without change to any of its code.

In short, the use of monitors in this manner reduces the software implementation effort for multiprocessor systems to the scope of single-processor designs. Or from another viewpoint, greater degrees of performance can be achieved from one basic design simply by adding processors.

IMPLEMENTATION NOTES

A run-time kernel is needed to manage concurrent programs according to the principles of Concurrent Pascal. The kernel is responsible for waiting and dispatching processes (sometimes called short-term scheduling), and essentially is a miniature executive. In this case, it also recognizes monitor program modules and provides exclusive access to their respective shared data structures. Additionally, the kernel manages *delay-queues* and implements the *delay* and *continue* operators.

Given the kernel, the systems programmer builds an application program by coding appropriate process and monitor modules that solve his problem. These program modules are then linked together and loaded into the computer with the kernel. These steps can be automated with a compiler. In fact, the existence of the kernel is transparent to a program written in Concurrent Pascal or equivalent language. (Concurrent Pascal also provides strong type checking and a notion of access rights, both of which are outside the scope of this paper.)

Space does not permit documenting a complete kernel, but Figure 2 is a basic flowchart of four key kernel routines. Illustrated are mechanisms for entering and exiting a monitor and for implementing the *delay* and *continue* operators. These routines are executed in behalf of the using program when requesting the services of the kernel. The DELAY and CONTINUE routines in Figure 2 adhere to the conventions of Concurrent Pascal; note that CONTINUE implies EXIT as well. Alternative kernel designs are possible, as deemed appropriate for given installation objectives and requirements.

Support for multiprocessors is reflected in Figure 2 in several ways. In particular, the kernels (one in each machine) share data structures located in shared memory to control their operations. They need a mechanism for resolving contention when more than one kernel tries to access a common kernel data structure. This can happen when selecting a process to dispatch and when attempting to grant access to a monitor for a process. A hardware read-modify-write function (Test and Set instruction) is employed to lock a data structure before a kernel accesses it. When a kernel finds a structure busy, it "spins" on the lock, testing it until it becomes free. Locking is depicted in Figure 2 by boxes with titles. The kernel need not necessarily treat a busy lock as a busy wait situation, but it is not likely to have anything else to do except spin.

Processor time consumed while spinning on a lock is data contention overhead and will degrade total system performance if the frequency of accessing a given structure is high among the processors. Probability of contention can be minimized by dedicating a separate lock for each monitor and for each process dispatch chain in the system. Although the lock intervals can become a bottleneck when average process execution time between kernel calls approaches the execution time of the kernel routines, this is generally not the case because the lock intervals are relatively short. For example, in Figure 2 the lock interval in the ENTER MONITOR routine is not for the duration of the monitor procedure, but rather only for the short interval the kernel takes to interrogate the monitor's busy flag and to chain the process to the monitor if the monitor is busy with another process. Upon exiting a process from a monitor, the kernel interrogates the monitor's busy chain and makes a waiting process (if any) ready for executing the monitor.

A process can be either dedicated or shared. The code for a shared process is located in shared memory and can be executed by any processor in the system; a dedicated process is located in a processor's private local memory. A shared process has an advantage over a dedicated process in that it can be dispatched by any processor in the system, whereas a dedicated process might be in a ready state but unavailable to idle processors in the system. While there are a number of reasons for having shared processes, they might execute slower than dedicated processes because of memory access contention and particular hardware constraints of a given installation. Processes might also have to be dedicated if they frequently access resources (peripherals, clocks, sensor equipment, etc.) that are accessible only on a given processor.

Process dispatching is distributed (except for the control structures in shared memory) because any processor's kernel can set processes ready for execution on other processors in the system. As illustrated in Figure 2, this can happen when a process waiting on a monitor busy chain is activated and when a process waiting in a *delay-queue* is continued. No single kernel is responsible, nor need be interrupted, for deciding the order of putting processes in a ready state. When a kernel selects a process for dispatching it selects the highest priority process that it is capable of executing (processes inside a monitor have priority over processes not in monitor mode).

Figure 2 reflects some design choices that would not be required for a single processor system. For example, when *continu*ing a process waiting in a *delay-queue,* the kernel might not be able to switch immediately to the *continue*d process because it might be dedicated to another processor. So the kernel routine falls through Dispatch instead. In fact, Dispatch is exercised whenever the kernel has the opportunity in order to insure fair priority treatment of processes in monitor mode. Otherwise, ready processes inside monitors might be blocking resources needed by other processes[6] and therefore idling processors.

An inter-processor interrupt facility enables a kernel in one processor to signal the kernel in another processor when setting one of its processes ready for execution. All idle processors can be alerted when readying a shared process. The interrupt need not convey any information except to cause the recipient kernel to cycle through its Dispatch routine. If processes are not to be preempted, which has merits germane with the philosophy of the monitor concept,[16,17] the interprocessor interrupt need be generated only when the destination processor is idle (or when the readied process is in monitor mode, if so desired to allow monitors to preempt processes). The interrupt facility is not an absolute requirement, however. When in idle mode with no

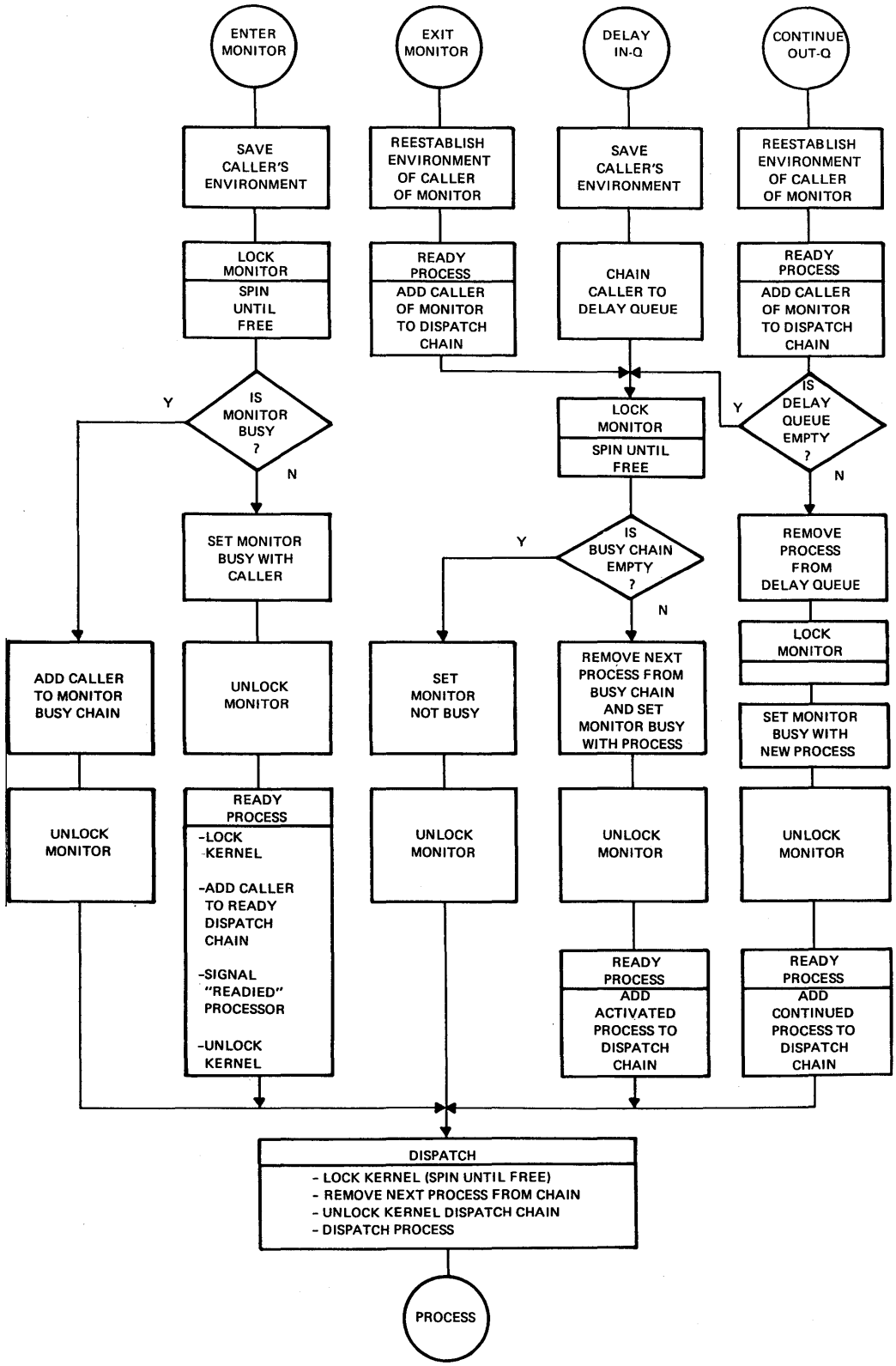


Figure 2—Kernel procedures to support monitors

work to perform, the kernel could continually cycle through its dispatch chains instead of going into a wait state (or perhaps periodically on a clock interval).

EXAMPLE SYSTEM

An example problem is presented to illustrate how monitors can be employed in building non-trivial concurrent programs. The utility of the monitor concept is evaluated by comparing the results with another solution of the same problem that did not employ monitors. Performance measurements are also given of various test runs when applying the monitor solution, as a multiprocessing program, to a dual processor configuration.

The following example application was defined as a benchmark model for investigating kernel designs and for evaluating relevant performance aspects. The benchmark is intended to reflect much of the character of a highly interactive transaction processing system, but actually it is a hypothetical application wherein data formats and processing functions are kept to a minimum (simulated) in order not to obscure the inherent interactions in the application problem being solved.

The benchmark application was implemented twice on an existing multi-tasking operating system in assembly language. Conventional operating system features for interprocess communications and synchronization were employed in the first implementation. This was done before any monitor capability was even designed. The second case employed monitors. A kernel facility employing the techniques described in the previous section was developed to run under the environment of an operating system task.

The application can be briefly described as terminal operators being able to exchange messages and to start concurrent job processes which execute disc I/O and computing functions according to the operator's request. The job programs send their results to the terminal specified in the originator's message. Job operations and terminal message exchanges run concurrently. Terminal I/O is overlapped so that an operator can input a message while the previous message is being processed. Output messages preempt input; job output takes presidence over terminal messages; job requests are serviced first-in-first-out. Each message delivered to a terminal contains a header identifying the source unless the previous output was from the same source.

Only the monitor approach to the benchmark problem is described here. Documenting the solution employing conventional techniques would serve no useful purpose. The monitor solution is itself somewhat arbitrary; there are probably many other ways of solving the benchmark problem in terms of monitors and processes. The approach taken here attempts to balance performance with conservation of memory. The benchmark program is built up in stages to simplify testing. Five terminals and two job processes are supported.

The benchmark program is described in pictorial form in Figures 3 through 5 with circles representing processes and boxes representing monitors. Program modules are identified by a global name and a type name in parentheses.

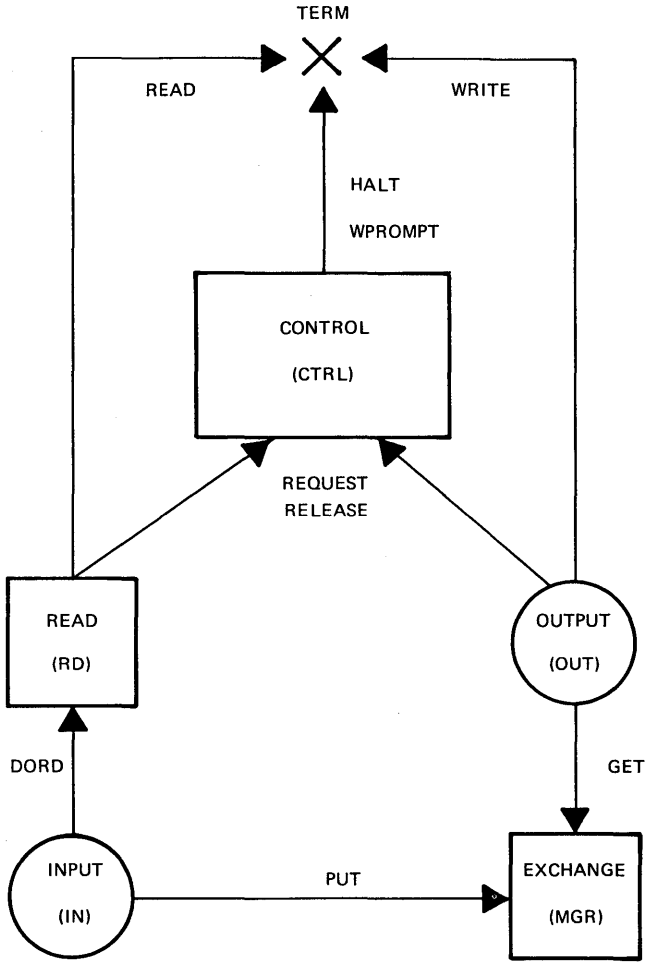


Figure 3—Benchmark terminal subsystem

Program modules of the same type have identical code; their data structures are replicated during program integration to build separate instances of the same module type.

The first stage consists of a single terminal subsystem as illustrated in Figure 3. The INPUT process continually inputs messages and passes them on to the message EXCHANGE monitor. The OUTPUT process continually gets messages from the EXCHANGE monitor when available and outputs them to the terminal. The terminal resource CONTROL monitor is responsible for granting exclusive access to the terminal and for maintaining its read/write/idle mode. The CONTROL monitor preempts a pending read I/O by executing halt I/O if the OUTPUT process requests access while the terminal is in read mode. The purpose of the READ monitor is to insure, by being a higher priority program module than the OUTPUT process, that the read I/O will be physically posted before the OUTPUT process can attempt to preempt it and cause contention problems with halt I/O. The CONTROL monitor also writes the prompt character while granting read access to avoid potential contention with halt I/O.

The next stage is to add the JOB processes as illustrated

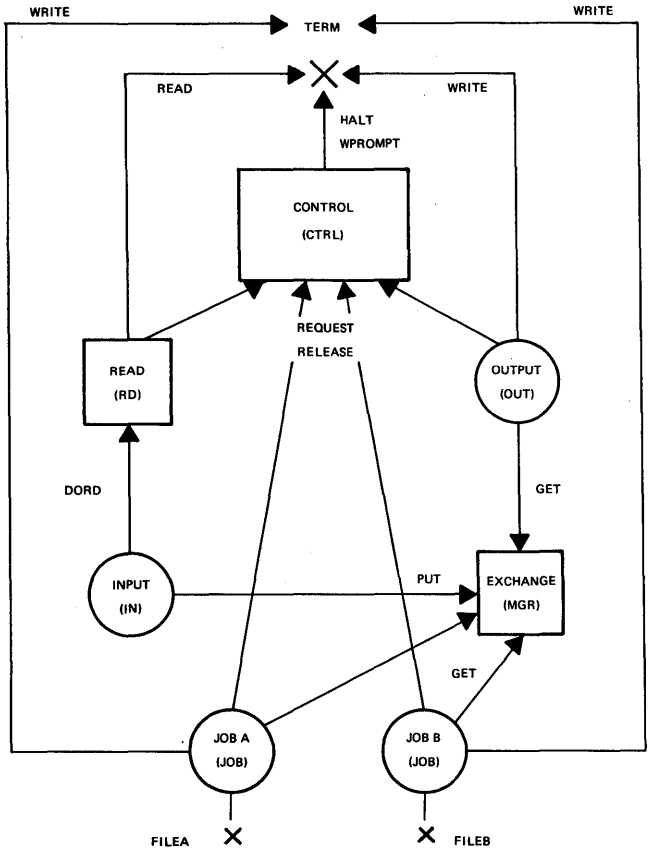


Figure 4—Single terminal benchmark system

in Figure 4. The aspects of controlling the JOB processes is much the same as for the OUTPUT process. However, with the addition of the JOB processes, the CONTROL monitor must maintain the identification of the last writer in order to indicate whether or not a header message is required on a new output request. It must also provide a delay queue for the JOB processes separate from the OUTPUT process in order to give priority to job output versus terminal output. Separate EXCHANGE monitors could have been provided for the JOB and OUTPUT processes, but a combined monitor approach was selected to conserve memory. Code in the EXCHANGE monitor is kept to a minimum to avoid contention losses in multiprocessor configurations.

Adding terminals to Figure 4 impacts the buffer requirements of the EXCHANGE monitor. It must provide an exchange mechanism for each source-destination pair, times two to allow overlapped I/O. This number in terms of buffers is not actually required by the application (in fact two times the number of terminals is adequate), so a buffer pool is employed in EXCHANGE to dispense message buffers, two per INPUT at most. With the addition of Pop and Push procedure calls in EXCHANGE for obtaining and releasing buffers, the full benchmark program is illustrated in Figure 5 (for five terminals and two job processes).

Although this third stage appears to be complex, it is actually a trivial extension. All complex logic (of CONTROL and EXCHANGE in particular) was verified in the previous two stages (CONTROL and EXCHANGE were initially written with all the features needed in the final stage).

The total application consists of:

- 5 input processes of type IN
- 5 output processes of type OUT
- 2 job processes of type JOB
- 5 read monitors of type RD
- 5 control monitors of type CTRL
- 1 exchange monitor of type MGR
- 5 logical terminals of type TERM
- 2 logical devices of type DISC
- 27 delay queues (not described)

Clearly the system is highly modular and structured.

Comparisons between the two implementation philosophies for a single processor configuration are summarized in Table I. Development effort for the kernel is not included in Table I, as it is a generalized facility taking the place of the tools otherwise provided by the conventional operating system.

Although the design effort was the same, coding and testing of the monitor solution took less time which resulted in nearly a doubling of productivity. Documentation, acceptance testing, and other activities normally encountered in a production environment are not included in these figures; however, the overall improvement should still be significant considering the understandability, modularity, and structured design of the monitor approach. While developing the monitor solution to the benchmark application, we experienced no aggravation in resolving race conditions and in defining correct inter-process communication protocols as occurred in the conventional case. The translation from design to code was also smoother, and the design itself was much more lucid. For example, when running tests to measure minimum buffering requirements we were unable to explain the conventional-system results, but the monitor-system results were predictable.

TABLE I—Benchmark Comparisons

	Conventional System	Monitor System
Development Time (work-days)		
Design	5	5
Code	11	5
Test	9	3
Total	25	13
Memory Size (1,000 bytes)		
Sharable code	3	3½
Total including data (monitor system includes over ½K for multiprocessing)	12¾	13¼ (note)
Performance Tests (messages per unit time)		
User transactions with minimum job processing	157	192
Execute-bound exchanges	273	416
I/O-bound exchanges	143	228

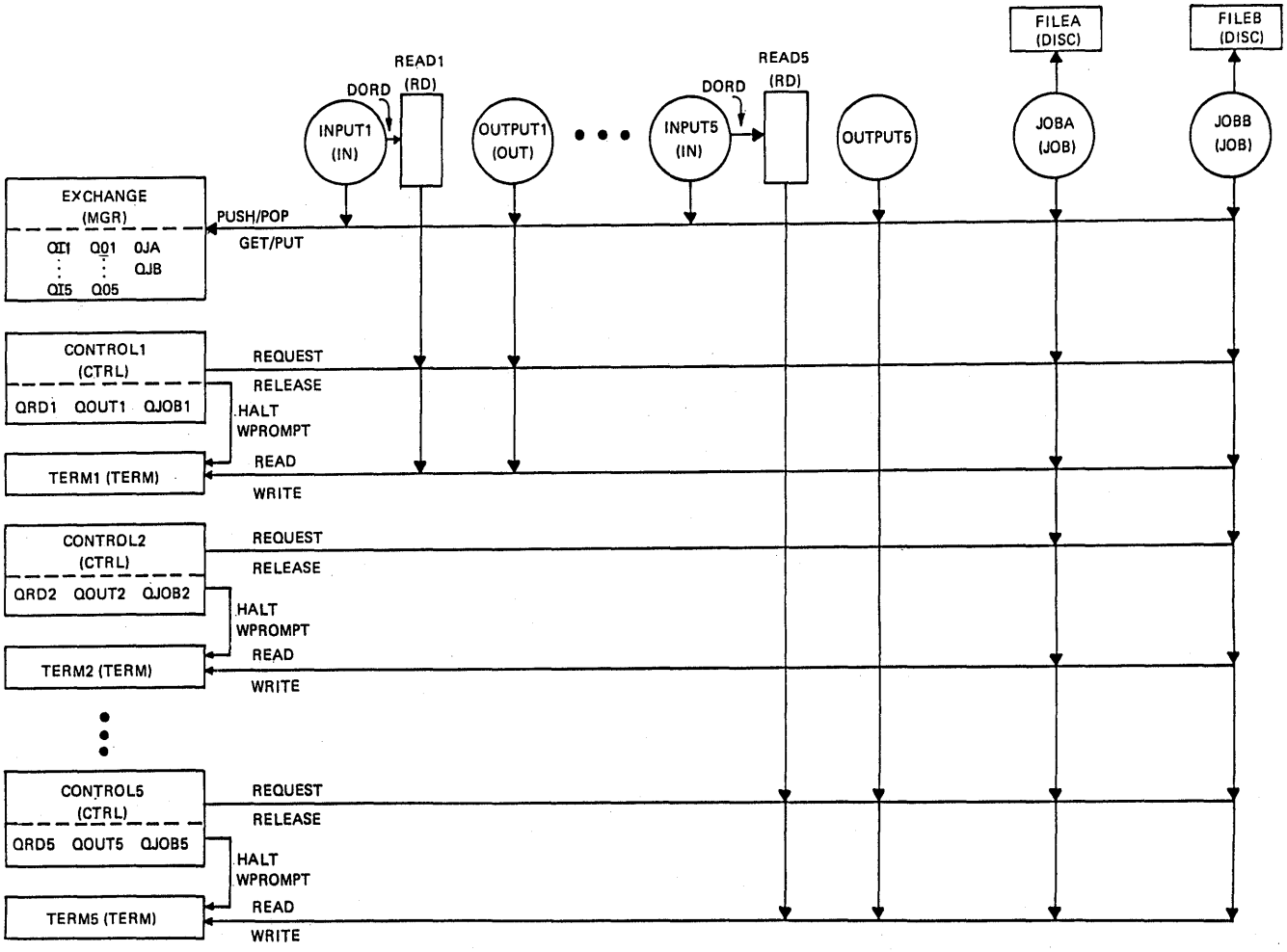


Figure 5—Benchmark system

The two solutions required approximately the same amount of memory. It should be noted that the kernel for the monitor solution was designed to support multiprocessing while the conventional-system was intended to run only on a single processor. The monitor solution therefore incurred unnecessary overhead when applied to a single processor.

The single processor monitor-system ran considerably faster than the conventional-system. The execute-bound test simulated maximum loading for terminal to terminal message exchanges. The I/O-bound test was performed with a single high-speed "terminal" (magnetic tape) sending to a low speed terminal. The better performance of the monitor solution, even with its resident multiprocessing overhead, is attributed to a better design and to more efficient interprocess communications with monitors versus existing operating system features.

The conventional-system was not designed to support multiprocessing because it seemed to be a very laborious and unproductive task. The real test of the monitor approach came about when we were able to bring up two different multiprocessor versions—a dual processor and a four-processor configuration with shared JOB processes—of the identical application program in only the short time needed to build load modules for each machine. After debugging the application on a single processor, *no errors* were uncovered during any of the multiprocessor tests on real hardware. Indeed, the processes which ran interleaved concurrently on a single processor ran in parallel on multiple processors *without change to any of the application code.*

Performance comparisons between single and dual processor configurations are given in Table II. In these tests the processors were the same model, and the data structures stored in shared memory for the dual processors were located in the same physical memory module for the single processor tests; furthermore, code was not executed out of shared memory. Consequently, memory access times were the same for the two configurations.

In the first test case, messages were exchanged between "null" terminals which did not involve real I/O nor much processing. The result is that the extra processor helped little (33 percent) because the application exhibited little

TABLE II—Multiprocessing Comparisons

	Single Processor	Dual Processor
	(Messages per unit time)	
Terminal-to-terminal exchanges with "null" devices	383	510
User transactions with job processing (per processor)	182	181
Concurrent execution of job transactions	182	357

parallelism in this test run and because data contention (spinning on locked data structures) was a bottleneck due to the very short execution periods of the application routines.

The second test represents a more realistic situation. The rate of processing user transactions was measured on the single processor. In the dual processor case, one processor was kept busy executing job computations while the second processor processed the same user transactions as in the single processor case. The dual processor was able to process essentially the same number of messages as the single processor (99 percent) along with heavy computing loads.

The third test was derived from a different, but similar, application. It allowed the compute functions of job processing to be executed in parallel. The dual processor was able to process 1.96 times as many messages per unit time as the single processor.

CONCLUSIONS

We conclude that the monitor concept is indeed an effective tool for building concurrent system programs. Furthermore, the concept can be applied as a useful design methodology, irrespective of the implementation mechanisms. Highly modular and structured systems will result with minimal opportunities for obscure time-dependent bugs and deadlock situations.

We applied the monitor concept to multiprocessing and demonstrated its particular utility for such applications. System programs intended to run on multiprocessor configurations can be reduced to the scope of single processor designs by employing monitors. We also illustrated techniques for implementing a run-time kernel to support multiprocessing. With such a kernel, integrated distributed processing programs can be built that accommodate different multiple processor configurations without change to the application code.

ACKNOWLEDGMENT

Messrs. G. Anderson, J. Goettelmann, T. Kibler, and D. Schmaltz are thanked for their helpful contributions and review.

REFERENCES

1. Enslow, Philip H. Jr., "Multiprocessor Organization—A Survey," *Computing Surveys,* Vol. 9, No. 1, March 1977, pp. 103-129.
2. Enslow, Philip H. Jr., *Multiprocessors and Parallel Processing,* John Wiley & Sons, 1974, p. 123.
3. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," *Communications of the ACM,* Vol. 17, No. 10, October 1974, pp. 549-557.
4. Brinch Hansen, P., "The Programming Language Concurrent Pascal," *IEEE Transactions on Software Engineering,* Vol. SE-1, No. 2, June 1975, pp. 199-207.
5. Brinch Hansen, P., *Concurrent Pascal Introduction,* Information Science, California Institute of Technology, July 1975.
6. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," *Operating Systems Techniques,* Academic Press, 1972, pp. 72-93.
7. Wirth, N., "Modula: A Language for Modular Multiprogramming," *Software-Practices and Experience,* Vol. 7, No. 1, January 1977, pp. 3-84.
8. Brinch Hansen, P., "Experience with Modular Concurrent Programming," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 2, March 1977, pp. 156-159.
9. Wallentine, V. and McBride, R., *Concurrent Pascal—A Tutorial,* Department of Computer Science, Kansas State University, November 1976.
10. Kessels, J. L. W., "An Alternative to Event Queues for Synchronization in Monitors," *Communications of the ACM,* Vol. 20, No. 7, July 1977, pp. 500-503.
11. Andrews, G. R., and McGraw, J. R., "Language Features for Process Interaction," *Proceedings of ACM Conference on Language Design for Reliable Software,* March 1977, pp. 114-127.
12. Lomet, D. B., "Process Structuring, Synchronization, and Recovery Using Atomic Actions," *Proceedings of ACM Conference on Language Design for Reliable Software,* March 1977, pp. 128-137.
13. Silberschatz, A., R. Kieburtz, and A. Bernstein, "Extending Concurrent Pascal to Allow Dynamic Resource Management," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 3, May 1977, pp. 210-217.
14. Howard, J., "Signalling in Monitors," *Proceedings of 2nd International Conference on Software Engineering,* October 1976, pp. 47-52.
15. Spencer, J. P. (Editor), *Distributed Systems,* Infotech State of the Art Report, Infotech International Ltd., Berkshire, England, 1976, p. 14.
16. Jensen, E. D., "Distributed Processing in a Real-Time Environment," *Distributed Systems,* Infotech Report, 1976, pp. 303-318.
17. Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel, and F. E. Heart, "Pluribus—A Reliable Multiprocessor," *Proceedings AFIPS 1975 National Computer Conference,* AFIPS Press, Montvale, New Jersey, pp. 551-559.

Data accessibility in structured programing

by NED CHAPIN
Data Processing Consultant
InfoSci Inc.
Menlo Park, California

INTRODUCTION

C. A. R. Hoare, N. Wirth, E. W. Dijkstra, and some others active in structured programing have described and tacitly recommended a hierarchical accessibility of passed data in structured programing (see for example, References 11 and 26). This fits well, they and others indicate, with both the theory and the practice of structured programing (see for example, References 15, 18, 19, and 25).

This paper does not question that the hierarchical accessibility with parameter passing can be made to work. It may even be the most elegant data accessibility for some interesting academic problems. However, this paper does argue that for most practical software design situations, other approaches are on balance superior, especially for the kinds of situations commonly implemented using COBOL, FORTRAN, PL/1, JOVIAL, or BASIC. To support this argument, this paper presents first some definitions of terms, and then an examination of some evidence in nine categories. Following a ranking evaluation, the paper draws a conclusion.

DEFINITIONS

For purposes here, a system or program designed and implemented with structured programing techniques is characterized as having a proper nesting of functions in three basic control patterns with single entrance and exit, in the sense defined by Mills.[20] This paper does not use "structured programing" to mean "structured coding," but rather uses it in the sense sketched by Dijkstra.[5] Further, structured programing is characterized by a hierarchical structure of modules (segments) again in the sense defined by Mills, which to people appear to specify the performance of specific single functions, as defined by Parnas and McGowan.[19,22] The function parsing techniques that lead to the creation of such proper-nested structures have been described elsewhere.[9] The object is ease of human understanding.[7,16]

For the purposes of this paper, a level of abstraction as it applies to data is defined to be a grouping of data based upon the recognition by people of some common characteristics, often semantic rather than structural.[11,19,24] Data conforming to levels of abstraction is referred to in this paper as "leveled data." For example, a file (composed of records in turn composed of fields) is regarded as more abstract than the component fields individually. In contrast, a disjoint string is defined to be a string data structure which has as member elements items of data with no necessity of similarity in level of abstraction. Thus a disjoint string might be composed of two Boolean variables, a record from a file, three tables, and a queue. Pointers may be among the elements of a disjoint string.[17] Because of the potential diversity, each member element in the string is specifically enumerated and described.

For the purposes of this paper, a flow graph is defined to be a graphic presentation of part or all of a control flow in a system or program.[1] The edges in the graph represent an explicit flow of control unaccompanied by the manipulation of data but accompanied by the implicit or explicit communication of data. The nodes in the graph represent an implicit flow of control accompanied by successive steps in the explicit manipulation of data, and accompanied by some means for the communication of data among the steps.

EXAMINATION

Access volume potential

The access volume potential is a count of the maximum possible number of data items with assigned values which are accessible to any given module upon invocation, summed for all modules. Ideally, this count should be small, since the larger it is, the greater is the difficulty of human understanding in design, implementation, and maintenance.

The calculation of the access to data items has been described by Allen and by Cocke.[1,2] Following "procedure A" as an "algorithm for finding intervals," a partition can be made of a control flow graph into basic intervals. If the hierarchy of modules is a true tree conforming to a proper nesting of functions, then it is possible to reduce selectively the basic control graph to yield nested intervals of higher and higher order, as diagramed in Figure 1. These nested intervals can correspond not only to the modules, but also to the levels in the hierarchy, since proper nesting in the hierarchy yields a reducible control flow graph.[13] Such a selectively applied flow-graph reduction is roughly equiva-

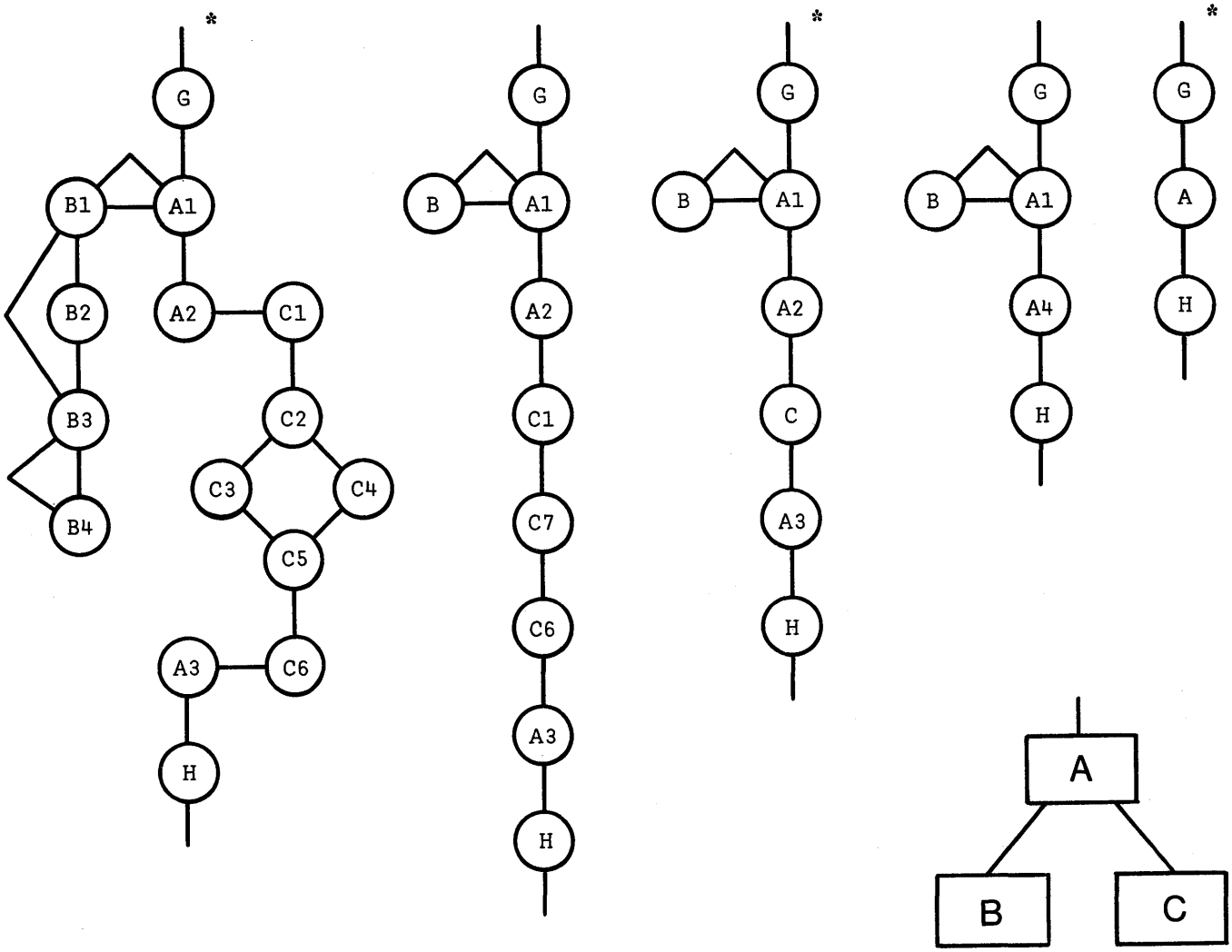


Figure 1—Successive (left to right) interval reduction in a control flow graph illustrating different module correspondence, as marked by *, and by use of identifying letters and subscripts, such as A2 (a part of Module A) etc.

lent to the reverse of a top-down application of function parsing guidelines.[9]

Because of this correspondence, the control flow graph can be used as a basis for determining the intrasystem access to data, given a structured programing design, and implementation. With global data, the "live" access potential (L) for any one module is given by the union of (a) the union (U) of the assignments (AN) and definitions (DN) of data within the module, with (b) the union of the data assignments and definitions available on the entrance edge of module i that do "reach" (R) that module:[1,2]

$$L_i = R_i U (AN_i U DN_i) \tag{1}$$

Note that the questions of whether or not the module i preserves or changes the definition of assignment, or whether or not data so specified are used in the module is irrelevant in determining the live access potential at any one module. If modules be identified in execution sequence, then the reach (R) expands as the control flow covers more modules. Thus, the access potential increases with each module executed by the new assignments AN and new definitions DN effective in each module.

If ρ indicates a count function, then the volume or amount of live access potential (A) in a program or system of n modules with global data communication is estimated by:

$$A = \sum_{i=1}^{n} \rho L_i \tag{2}$$

With passed data, the amount of live access potential is much smaller than for the global case with the use of structured programing techniques. The live access potential is given by expression (1) as before, but now the set R_i is much reduced because it is a subset of the global data case, limited to the data deliberately communicated between the i^{th} modules and other modules. This also keeps expression (2) much smaller.

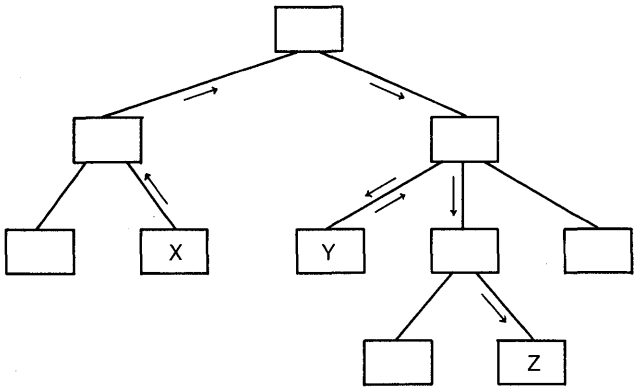


Figure 2—Illustration of data pass-through and pass-through chains

With leveled data, the minimum total number of data items assigned values (AN) or defined (DN) is never smaller, and is usually larger, than the corresponding values for the disjoint strings. This is because the disjoint strings make provision only for data items actually accessed, whereas leveled data may include data defined or assigned for consistency, elegance, or symmetry of conception.

Expressed as a ranking, with 1 for the most desirable, and 4 for the least, passed disjoint strings are of rank 1, and global leveled data are of rank 4. This is summarized as the top row in Table I.

Access values

The access to data values treats each item of data as a different item of data when it takes on a different value. A count of these is an indicator of the diffuseness of function in the hierarchy and of the difficulty of debugging and maintaining a system of program.

The estimating procedures for access to values *used* for global data has been well presented by Allen and Cocke.[2] Domains are difficult to specify from the flow graph and from the hierarchy.[10] In practice, the availability of input-output tables is helpful.[7]

With passed data, the access to data values is much restricted, and domains for data entering a node from along any edge of the control flow graph can be cleanly specified. In terms of data values *used* in the module, the access is the same as for the global case noted above. But to this must be added the access to values arising from the communication of the data through the module. Much data may have to be passed through some modules unchanged in value or identity in a hierarchical structure. Diagramed in Figure 2 is a situation where an item of data is assigned an identity and value in module X and accessed in module Y to be modified in value but not identity for use in module Z. The communication from X to Y involves a pass-through chain containing three pass-through nodes, and from Y to Z of two pass-through nodes.

With leveled data, the count of the data values accessed is greater than for the disjoint string, because extraneous values are presented in a module at levels above the lowest in the hierarchy.[23] Hence, the ranking for access to values as summarized in Table I is best for the global disjoint string, and worst for the passed leveled data.

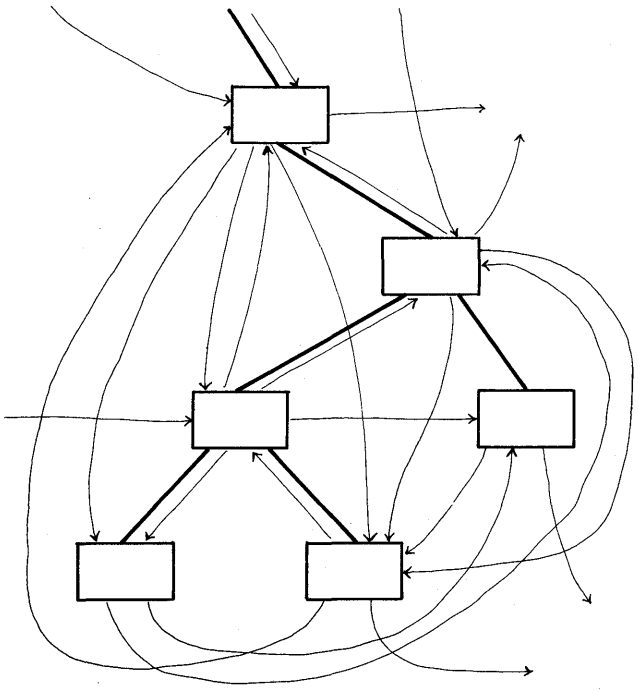


Figure 3—Illustration of weak congruence between control flow marked as heavy lines, and data flow marked as light lines

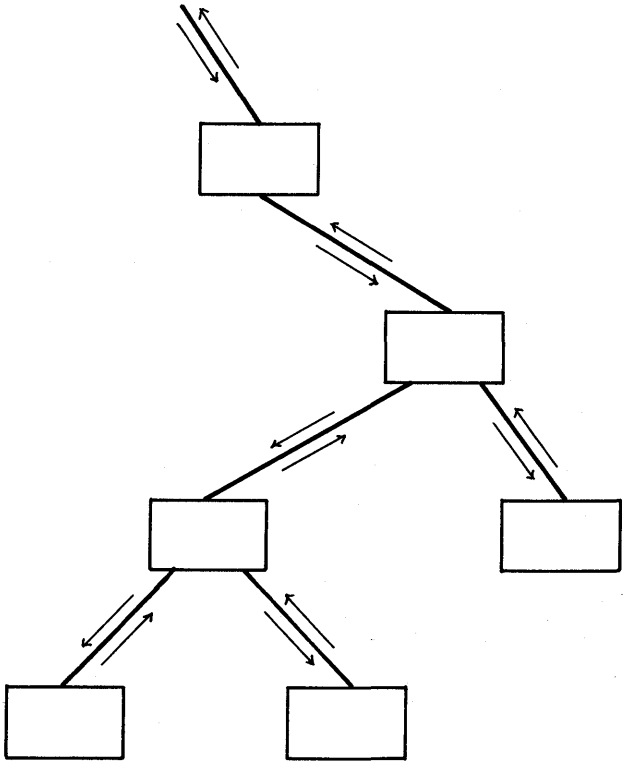


Figure 4—Illustration of strong congruence between control flow and data flow

Congruence

When the pattern of the data communication among the modules of the hierarchy has a high degree of congruence with the pattern of the flow of control, design, implementation, and maintenance work is easier. People only have to know one pattern to understand two aspects of the program or system. The use of input-output tables helps reveal the degree of congruence between data and control flow.[7]

Global data offers little congruence with the nested hierarchy of the tree, as diagramed in Figure 3. This is the case for both leveled data and disjoint strings. This is because the data flow is not constrained to match the control flow, and in practice is usually a network, not a tree if the modules be treated as nodes for data flow.[14]

Passed data offers excellent congruence with the nested hierarchy of the tree, as diagramed in Figure 4. This is the case for both leveled data and disjoint strings, but the disjoint string reduces the presence of unused data. The ranking, therefore for congruence places the passed disjoint string in the best position, and gives a tie for the two alternatives in the global data case, as shown in Table I.

Coupling

While the concept of coupling comes from structured design, it can be applied to software designs done with structured programing. Using the definition and scale of coupling as laid out by Myers, the global data case is at best common coupled.[21] This scale position is the same for both leveled data and for disjoint strings.

The passed data case requires a closer inspection. Using leveled data results in stamp coupling except sometimes at the leaf position in the hierarchy. Using disjoint strings enables the consistent use of elementary items of data throughout the hierarchy, and results in the more desirable data coupling between modules on adjacent levels vertically in the hierarchy. In either situation, no coupling at all characterizes all of the other coupling relationships between modules. In short, the ranking of coupling is the same as for congruence, as noted in Table I.

Strength

While the concept of strength comes from structured design, it can be applied to software designs done with structured programing. Using the definition and scale of strength as laid out by Myers, the evidence on strength is moot.[21] Sometimes one alternative seems to go with higher strength and sometimes not. Hence, the ranking for strength is a tie for all, as shown in Table I.

While not evidence, an observation may add insight. Since in structured programing, the focus is on the function of a module, the data flow into and from a module serves effectively as a definition of that function. For high strength, a module must require all of its input to yield its outputs, and not be partitionable into two or more parts with no increase in the number of intermodule items of data needed, ignoring pass-throughs. Therefore, a partitioning of functions implies a partitioning also of data flow if high strength is to be maintained, as diagramed in Figure 5.[6,8] The means of achieving that data flow is really an independent question. In a high-strength design, the actual access of data for use is generally closely similar whether the design or the implementation be passed data or global, but the "pass-through unchanged" pattern is very different.

Overhead

Execution overhead in an implemented program or system depends on many factors, including the computer configuration, the language, the compiler, the operating system, the shop practices, etc. Even if attention be given only to two aspects, such as CPU cycles used, and amount of internal storage committed, only general trends can be described. No firm consistent evidence is available, or probably even possible beyond an instance by instance basis.

Global data are the most economical of storage space and fastest in execution speed if disjoint strings be handled. Both measures are impaired by using levels of abstraction because of the increased size of the units of data handled. This is reflected in the rankings shown in Table I.

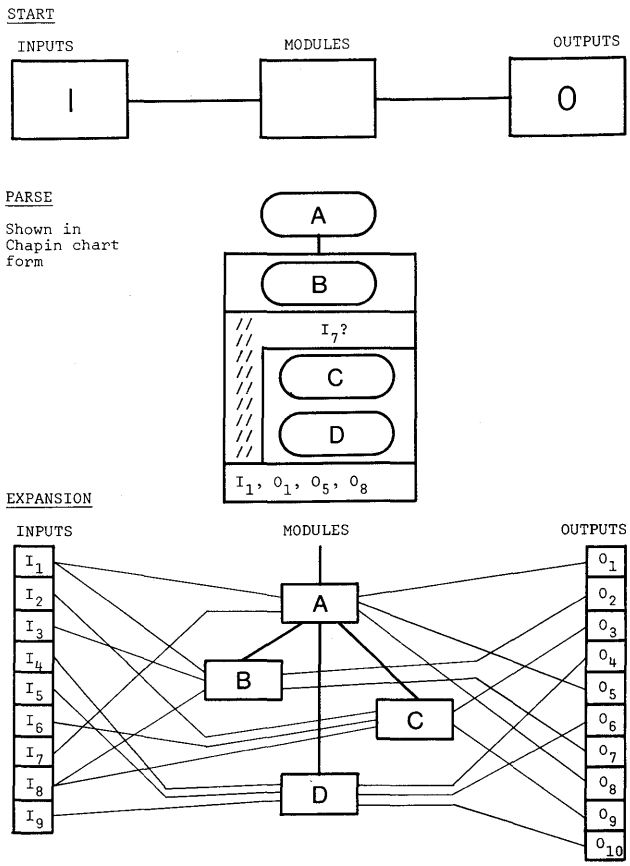


Figure 5—Diagram of some data implications of high strength

Passed data are still less economical and slower, again in the same two steps, because of the increased handling of data that are involved. The higher the strength of the modules, the smaller is the impairment because the more elegant is the use of data. But the smaller the modules the larger is the impairment, because calling and passing becomes a larger proportion of the total work to be done. This degrades the ranking for the passed data alternatives, as shown in Table I.

Human convenience

Both the global and the passed data are convenient for people, but in different ways. The global data offers single description (declaration), economy of effort, and ease of reference to the description for people. But global data also offer the least visibility into the steps in the use, transformation, and production of the values of the data—a critical matter in a function-oriented approach.

The passed data offer uniformity in conventions and high visibility. But the cost is in redundancy and repetition, both in the designs and in the code. At the compile stage the language conventions provide an automatic check on human performance. This enforcement helps keep the data visibility high.

Convenience is also both aided and hurt both by leveled data and by disjoint strings. The leveled data provide an easily comprehended grouping of data but hide data identities—a serious matter in debugging and maintenance. The disjoint strings lack the mnemonic quality that aids the human memory, but do focus attention on actual functions.

A ranking of the human convenience of necessity is subjective, and reflects value judgments about operating environments as well. An informal survey conducted over half a year's time involving persons who design computer software and try to use structured programing techniques, and including people from all across the continent, indicated the preferences shown in Table I on convenience.

Maintainability

Provided the visibility is good, the reports from people doing the work seems consistent on maintainability: the passed data in disjoint strings has the lowest cost of maintenance. Some of this comes from the access measures noted earlier, some from congruence, and some from coupling. People can follow the items of data more easily, and the functions are usually more clearly specified and understandable with the passed data in disjoint strings. By contrast, the leveled data have to be peeled away to specifics, and hence involve more maintenance effort.

For these same reasons, the global data offer less aid in maintenance. The value changes in the data are more difficult to follow, especially when leveled data are used. In the absence of countermeasures, it takes people longer to find out what is going on. Hence, in the summary ranking, the passing of disjoint strings is shown as a 1, and the leveled global data is shown as a 4.

System quality

System quality has many measures. One is number of bugs encountered over some period of time. Using techniques described by Haney and Myers, it is possible to give a quantitative tinge to measuring system quality in terms of bugs.[12,21]

Applying those measures to examples drawn from industrial and governmental computer applications, indicates that passed data in disjoint strings has the lowest expectation of bugs. Passed leveled data has an expectation about one-third higher. Global data in either form yield an expectation for nearly double the number of bugs. This gives a tie in the ranking shown in Table I.

Criticism of these techniques and the use of bug counts as quality measures has been heard. Both are admittedly imprecise—but they are also the best quantitative measures currently available. Some of the quantitative observations offered by Brooks are not inconsistent with the rankings shown in Table I for system quality.[4]

EVALUATION

A review of the evidence indicates a strong leaning in one direction as the Overall summarizes in Table I. The availability of this evidence gives the opportunity to take a fresh look at some alternatives and make rational choices to fit given situations and needs. Let us consider local, internal, and external data, for example.

Local variables are items of data made available in the design and implementation to only selected modules. If they are accessible only in the modules in which they are declared or defined, then they are true internal variables. If they are accessible to other modules, then all of those modules must be in the same branch of the tree that represents the nested hierarchical relationship among the functions. Effectively, the locals thus used become "restricted" globals. Local variables are well regarded in the theory of structured programing.[19,27]

Used with global data, local data are a step in the direction of passed data. Used with passed data, local data offer a relaxation of the visibility enforced with passed data. But the use of true internals degrades neither passed data nor global data usage. Internal data use may even enhance either but especially global data.

"External" data are another possibility. These are items of data communicated between specific modules anywhere in the hierarchy—not just between immediate superiors and subordinates. It is like a private line for communication external to and addition to those provided by the passing of data.

External data are better regarded in structured design than global data because they are more visible to people.[21] But their flow does not follow the tree, and as they are usually

TABLE I.—The Most Favorable Rank is 1, the Least Favorable is 4.

SYNOPSIS OF RANKING

Evidence	Passed Data		Global Data	
Factor	Leveled	String	Leveled	String
Volume	2	1	4	3
Values	4	3	2	1
Congruence	2	1	3.5	3.5
Coupling	2	1	3.5	3.5
Strength	2.5	2.5	2.5	2.5
Overhead	4	3	2	1
Convenience	2	1	4	3
Maintainability	2	1	4	3
Quality	2	1	3.5	3.5
OVERALL	2	1	4	3

used, they provide another (sometimes complicating) data flow for people to understand simultaneously with either passed or global data. A common use of external data accessibility is the use of scratch files to communicate data between selected modules, often relatively high in the tree.

Is there some form of intrasystem data communication that would rate better than any of those discussed thus far? Consider the following possibility, termed "monitored data," that combines the strong features of the forms and varieties already discussed.

1. Define or declare with a unique name each and every data item only once in the software system to be created. Incorporate in the definitions or declarations any list-linking or component or group-element relationships among the items of data.
2. Enable by a specific visible means in the documentation for the module, each module that may access the value of any given item of data or may assign the value, on an item by item basis, noting the role of the item of data. The roles may be for control, for processing, for modification of value, or for communication through unchanged.
3. Record and associate with the definition or declaration, the identification of the accessing and assigning modules and the roles of the data in those modules. This documentation, while not intrinsic to monitored data, adds substantially to the human convenience and to the maintainability.
4. Disenable all other modules for all items of data by a means that prevents their implementation if any access is explicitly or implicitly attempted in or by them to any item of data for which they have not been specifically enabled.
5. Make any access to any item of data be to *all* parts of that item of data (for example, enabling for a 1000-byte record yet only using a 2 byte field in it would be illegal) except where explicit decisions result in alternative access.

Some comparison of monitored data with the forms and varieties considered earlier may be interesting (see Table I). For systems and programs involving more than about 50 intermodule items of data, on volume, congruence, coupling, and strength, the monitored data alternative ranks the same as the passed-string alternative. On values and overhead, it ranks the same as the global-string alternative. On system quality, it ranks better than the passed-string alternative, by close to a factor of two. For systems and programs of smaller size and for some strongly mathematics oriented systems, the passed-string alternatives appear to be superior to the use of monitored data. But from the limited experience thus far, for most non-trivial systems and programs implemented in COBOL, FORTRAN, PL/1, JOVIAL, and BASIC, the monitored data appear to be the superior alternative.

Insufficient experience is available to give a confident ranking on convenience and maintainability. Some observations may be indicative, however. On convenience, the monitored data appear to offer all the advantages of both the passed and the global alternatives, and avoid the low-visibility of the global and some of the redundancy of the passed data alternatives. The verification of the enabling in points # 2 and # 4 requires either a manual or a software step. An automatic software step improves the human convenience. The usefulness and precedent for such steps already exists in some structured programing practices using precompilers, data base administrators, librarians, and project files.[3] Input-output tables can also be a useful aid.[7] Because of the excellent visibility it produces, the monitored data alternative appears to offer a maintainability potential superior to that of the passed-string alternative.

CONCLUSION

Among the four alternatives considered, the weight of the evidence favors the passed form of intrasystem data communication, especially where the overhead is not of serious concern. The disjoint string ranked superior or equal to the use of data in levels of abstraction on all nine factors considered. Except on convenience and maintainability where little evidence exists yet, the monitored form of intrasystem data communication appeared to be superior or equal to the passed-string alternative.

REFERENCES

1. Allen, Frances E., "Control Flow-Analysis," *SIGPLAN Notices*, Vol. 5, No. 7, July 1970, pp. 1–19.
2. Allen, Frances E. and John Cocke, "A Program Data Flow Analysis Procedure," *Communications of the ACM*, Vol. 19, No. 3, March 1976, pp. 137–147.
3. Baker, F. Terry, "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, Vol. 11, No. 1, 1972, pp. 99–117.
4. Brooks, Frederick P. Jr., *The Mythical Man-Month: Essays on Software Engineering*, Reading, MA: Addison-Wesley, 1974, 192 pp.
5. Buxton, J. M., Peter Naur and Brian Randell, *Software Engineering Concepts and Techniques*, New York: Petrocelli/Charter, 1976, 306 pp.
6. Chapin, Ned, "A Deeper Look at Data," *Proceedings of 23rd National Conference of the Association for Computing Machinery*, New York: Petrocelli/Charter, 1968, pp. 631–638.
7. Chapin, Ned, "Input-Output Tables in Structured Design." *Structured Design*, Maidenhead, UK: Infotech, 1978, in press.

8. Chapin, Ned, "New Format for Flowcharts," *Software Practice and Experience,* Vol. 4, No. 4, Oct–Dec. 1974, pp. 341–357.
9. Chapin, Ned, "Function Parsing in Structured Design," *Structured Design,* Maidenhead, UK: Infotech, 1978, in press.
10. Cocke, John, "Global Common Subexpression Elimination," *SIGPLAN Notices,* Vol. 5, No. 7, July 1970, pp. 20–24.
11. Dahl, O. J., E. W. Dijkstra and C. A. R. Hoare, *Structured Programming,* New York: Academic Press, 1972, 220 pp.
12. Haney, Frederick M., "Module Connection Analysis—A Tool for Scheduling Software Debugging Activities," *Proceedings FJCC 1972,* Montvale, NJ: AFIPS Press, 1972, pp. 173–179.
13. Hecht, M. S. and J. D. Ullman, "Characterizations of Reducible Flow Graphs," *Journal of the Association for Computing Machinery,* Vol. 21, No. 3, July 1974, pp. 367–375.
14. Hoare, C. A. R., "Data Reliability," *Proceedings 1975 International Conference on Reliable Software,* also published as *SIGPLAN Notices,* Vol. 10, No. 6, June 1975, pp. 528–533.
15. Horning, J. J., "Some Desirable Properties of Data Abstraction Facilities," FDT, Vol. 8, No. 2, March 1976, pp. 60–62.
16. Inglis, J., "The True History of Unstructured Programming," *Computer Bulletin,* Series 2, No. 5, September 1975, pp. 18–19.
17. Kieburtz, Richard B., "Programming Without Pointer Variables," *FDT,* Vol. 8, No. 2, March 1976, pp. 95–107.
18. Linden, Theodore A., "The Use of Abstract Data Types to Simplify Program Modifications," *FDT,* Vol. 8, No. 2, March 1976, pp. 12–23.
19. McGowan, Clement L. and John R. Kelly, *Top-down Structured Programming Techniques,* New York: Petrocelli/Charter, 1975, 288 pp.
20. Mills, Harlan D., *Mathematical Foundations for Structured Programming—Report FSC-72-6012,* Gaithersburg, MD:IBM Federal Systems Division, 1972, 62 pp.
21. Myers, Glenford J., *Reliable Software Through Composite Design,* New York: Mason and Lipscomb, 1975, 159 pp.
22. Parnas, David L., "On the Criteria to be used in Decomposing Systems into Modules," *Communications of the ACM,* Vol. 15, No. 12, December 1972, pp. 1053–1058.
23. Sanfield, Stuart H., "The Scope of Variable Concept: The Key to Structured Programming?" *SIGPLAN Notices,* Vol. 13, No. 7, July 1974, pp. 22–29.
24. Wirth, Niklaus, *Algorithms + Data Structures = Programs,* Englewood Cliffs, NJ: Prentice-Hall, 1976, 366 pp.
25. Wirth, Niklaus, "On the Composition of Well-Structured Programs," *Computing Surveys,* Vol. 6, No. 4, December 1974, pp. 247–259.
26. Wirth, Niklaus, "Program Development by Stepwise Refinement," *Communications of the ACM,* Vol. 14, No. 4, April 1971, pp. 221–227.
27. Wulf, William and Mary Shaw, "Global Variable Considered Harmful," *SIGPLAN Notices,* Vol. 8, No. 2, February 1973, pp. 28–34.

Program complexity using hierarchical abstract computers*

by WILLIAM G. BAIL

University of Maryland
College Park, Maryland

and

Automation Industries—Vitro Laboratories Division
Silver Spring, Maryland

and

MARVIN V. ZELKOWITZ

University of Maryland
College Park, Maryland

and

National Bureau of Standards
Washington, DC

INTRODUCTION

There is currently interest in measuring the complexity of a computer program with the evaluation of a program's control structure via its flowgraph representation.[1,2] However, the inclusion of the effects of data on these measures is often lacking. This paper proposes a new measure of program complexity that attempts to identify regions of locality by combining a control structure approach with information about data usage to get a valid measure of overall program complexity. The need for objective measures of program structure is important if the area of programming language design and implementation is to take on a more formal basis from the ad hoc techniques currently used.

This work is related to that of Hellerman,[3] Savage,[4,5] and Chaitin[6] in that the complexity of an algorithm is defined as the number of bits in the algorithm's representation, i.e., the number of bits in a program implementing the algorithm. A program implementing a given algorithm is considered to be better than another program implementing the same algorithm if it has fewer bits in its representation. In order to provide a standardized representation for algorithms, the concept of a Hierarchical Abstract Computer is defined (HAC). Based on a primitive-level HAC, hierarchical levels of HAC modules are used to specify the structures and sequences of a specific algorithm. The modules are chosen according to the concept of a Prime Program Parse,[7] which defines a particular, unique sequence of subroutine decompositions from an original flowgraph representation of an algorithm. The complexity of the total program is defined to be the sum of complexities of each of the modules used to form the hierarchy.

The difficulty of writing program code to implement an algorithm has been related to the amount of information that must be manipulated in the process of writing the program. There have been many attempts recently to simplify the process, most conspicuously aimed at the control structure complexities. Structured programming has been expounded as a method by which programs can be written with minimum difficulty, and mainly involves restraints on the use of control structures. There are also some methods which are directed at data structure complexities, such as the concepts of data abstraction and information hiding. These methods are concerned with the simplification of the data that is manipulated by the program code. By reducing the data available for referencing to only that which is directly needed by a section of code, the possibility for confusion is reduced. As much as possible, references to data are maintained in small sections of code, and are kept exclusive, in that only those sections are allowed to act on the data. In this manner, local data responsibilities are easy to isolate and to maintain. These local responsibilities are known as clusters, and their exclusiveness and privacy is known as information hiding.

It is assumed that a program is well-structured if clusters of data and control activity are small. The resultant small subroutines are thus accompanied by correspondingly small data spaces. The purpose of this research is to be able to measure different implementations of the same algorithm so as to determine the particular implementation which is best in the sense of being well-structured or least complex. The next section discusses some relationships between an algo-

* Research supported in part by Air Force Office of Scientific Research grant 77-3181 to the University of Maryland.

rithm and the set of programs which implement the algorithm.

ALGORITHMS AND PROGRAMS

An algorithm has been defined to be a finite set of rules which specify a sequence of operations used to solve some specific problem. The operations specified by the rules are selected from a set Ω of operators used by the algorithm. The proper sequencing of these operations depends on the state of the data items used by the algorithm, and is described with the use of a meta-algorithm or program which directly implements the algorithm. The essential difference between an algorithm and its implementing program is the specification of the operation sequencing through the use of control statements. Each algorithm has an infinite number of programs which implement it, some of which are considered to be well-structured, or less complex. It is this class of programs which a complexity measure attempts to identify.

Algorithms are rarely completely specified, in that assumptions are made about the characteristics of the data types and the operators which manipulate the data. Yet each operator is itself described by an algorithm at a lower level, and so on, down to some indivisible, lowest level set of operators and data types. Computer programs are written in high-level languages whose operators are implemented in machine languages, whose operators are in turn implemented in micro-code, whose operators are implemented in hardware gates, etc. The next section proposes a primitive set of operators and data types to be used as a basis upon which complete specifications of algorithms can be hierarchically built.

Rarely, however, does the operator set of the algorithm match the operator set of the programming language. A central problem in programming is identifying and building the inherent operators and data types that make up the algorithm, using the available, lower-level tools. Usually, the algorithm description given to the programmer does itself not adequately recognize the structures within the algorithm specified, and so, severely complicates the problem facing the programmer. His task is to perform certain transformations to the algorithm specification without altering its basic structure so as to implement the request as simply as possible.

Let Δ be the set of data items used by an algorithm or program, and let D be a subset of Δ. The *Execution Sequence* of an algorithm relative to D is defined to be the sequence of states assumed by the members of D.

Two programs are *execution equivalent* relative to D if they have the same execution sequence for all inputs. Different programs can be execution equivalent because the occurrence of control operations is ignored, having no effect on the execution sequence.

Two programs are *functionally equivalent* if, given the same input values, they produce the same output values.

Two programs are *semi-execution equivalent* relative to D if, for each $d \in D$, the programs are execution equivalent relative to d. That is, each data item of interest takes on the same set of values, but not necessarily all in the same relative order as the other data items.

It is a trivial result that if two programs are semi-execution equivalent, then they are also functionally equivalent. Functional equivalence is the weakest restriction, in that there need be no similarity in the two algorithms used by the programs. Execution equivalence is the strongest restriction, requiring two programs to have identical data transformations. The restriction of semi-execution equivalence is felt to be a reasonable criteria for comparing different programs, in that the implemented algorithms are not only functionally equivalent, but also structurally similar, in that all of their data items are undergoing similar transformations. There is a certain latitude available to the programmer in this situation to strive towards a simple implementation.

HIERARCHICAL ABSTRACT COMPUTERS

The concept of a set of programs interacting at different levels, and working together to implement an algorithm, leads naturally to the following definition of a Hierarchical Abstract Computer (HAC).

Def: A Hierarchical Abstract Computer (HAC) is a quadruple (Δ, I, Ω, T) where Δ is a set of distinct named data items, each of which is associated with a member of T, the set of legal data types, I is a set of named instructions, and Ω is a finite set of operation codes. Each instruction is of the form:

$$\omega d_1 d_2 \ldots d_n x_1 x_2 \ldots x_m$$

where $\omega \in \Omega$, $d_i \in \Delta$ and x_i is the name of an instruction in I, or $x_i = 0$. The d_i's are the data arguments for the operator ω, and the x_i's represent the possible alternatives for the next instruction to be executed. The data items and the instruction addresses (names) are numbered beginning with 0, and a transfer to address 0 means to halt execution for that HAC.

Operation codes are either composite or primitive. A composite code is itself implemented by a lower-level HAC, carrying out some specific algorithm on the input data. A primitive operation code is, in some sense, indivisible, and is defined to be one of the three operators:

TEST d x_1 x_2
SET d x_1
CLEAR d x_1

where $d \in \Delta$, and is associated with the data type BIT, having the value 0 or 1, and $x_1, x_2 \in I$. These instructions have the semantics:

TEST	If the contents of $d=0$, then the next instruction is to be found at address x_1; otherwise, the next instruction is at x_2.
SET	Set the data item d equal to 1.
CLEAR	Set the data item d equal to 0.

These three codes form the Primitive Basis, Ω_p, and, if

a HAC consists of Ω_p together with the data type $T=\{BIT\}$, then the HAC is said to be a Primitive HAC, or PHAC. An important characteristic of Ω_p is that it is complete, that is, it can compute all finite functions $f: \{0,1\}^n \rightarrow \{0,1\}$. This result follows directly from the PHAC implementation of the basic Boolean functions AND, OR, and NOT.

As described, a HAC program can be decomposed into lower and lower level modules until only Primitive HACs are obtained, thus forming a tree of modules. Conversely, given a PHAC implementation of an algorithm, sections of code can be removed and replaced by a single invocation to a new composite operator, with as many parameters as there were global variables across the code segment.

While a HAC is only as powerful as a finite state automata, it is still sufficiently powerful to model real computer programs that do not use auxiliary storage, and, unlike finite state models, can easily be measured with a complexity measure. This model can, however, be easily extended to include tapes. As in real hardware, specific addresses for data, control, and status registers can be defined resulting in a memory mapped I/O. Placing information into these registers causes information to be read or written from a tape and to appear in the data register. Thus storage may actually be infinite, but accessible only via a finite set of specific addresses.

PROGRAM COMPLEXITY

Given a HAC implementation of an algorithm, it is now necessary to define a complexity measure on this set of modules. The complexity of a HAC H is defined to be the sum of the complexities of each of the instructions in the HAC:

$$C(H)=\sum_{j\in I} C(j)$$

Let $L_\Delta=\log_2|\Delta|$=number of bits needed to address the data space,
$L_I=\log_2|I|+1$=number of bits needed to address the instruction space I (with the addition of address zero to be used as a STOP instruction), and
$L_\Omega=\log_2|\Omega|$=number of bits needed to specify the operation code.

Then, the complexity of an instruction i with m data arguments and n addresses is:

$$C(i)=L_\Omega+m*L_\Delta+n*L_1$$

For PHAC programs, the complexity of each TEST instruction is $(1.58+L_\Delta+2*L_1)$ and the complexity of each SET and CLEAR instruction is $(1.58+L_\Delta+L_1)$.

Thus, the complexity of a HAC implementation of an algorithm is defined to be the sum of the complexities of each of the HAC modules which form the implementation. For each program, however, there are many possible modularizations, any one of which could be chosen by the programmer, but which may not reflect the structure of the program itself. It is desirable to have a canonical modularization for a program to avoid this problem. This representation should be unique for a given program, and should correlate closely with the structural properties of the program. The Prime Program Parse meets these criteria.[7]

A *proper program* is a flowgraph with a single input arc, a single output arc, and with each node in the flowgraph being on a path from the input arc to the output arc. A *prime program* is recursively defined as a proper program with no proper prime subprograms of greater than one node. The process of identifying the hierarchy of prime programs which make up a compound program is known as a prime program parse. Such a parse is unique up to associativity of sequences of statements.

PROPERTIES OF MEASURE

This measure has been used to measure the complexity of several programs implementing common algorithms. In one example, an 8-bit adder was programmed on a primitive HAC, using 78 instructions, and resulting in a complexity of 918.26. The prime program parse of the program yielded 8 subroutines, all invoked by a single controlling routine. This program was restructured by forming three composite operators, resulting in a complexity of 471.76. This savings in

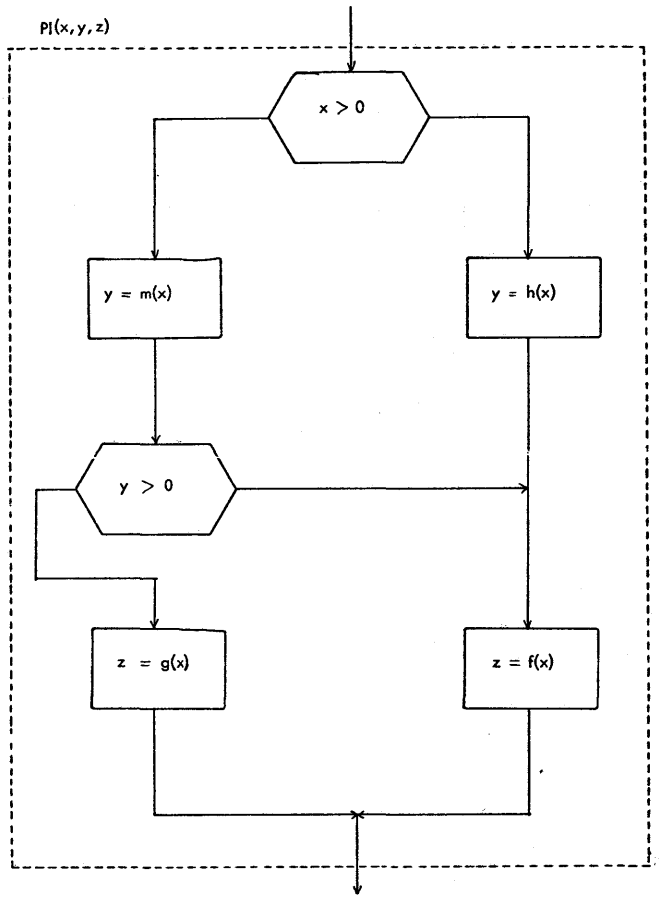


Figure 1—Unstructured version of program

complexity was due primarily to the recognition that 6 of the 8 modules in the prime program parse performed the same function, and thus represented a large degree of redundancy. The two programs are execution equivalent.

As an example of the comparison of two programs which are semi-execution equivalent, consider those shown in Figures 1 and 2. Figure 1 illustrates a nonstructured program which has a complexity of 53.16, as determined from the prime-program parse. Figure 2 illustrates a structured program, having a complexity of 52.02, and consisting entirely of small (size 1 or 2) prime programs.

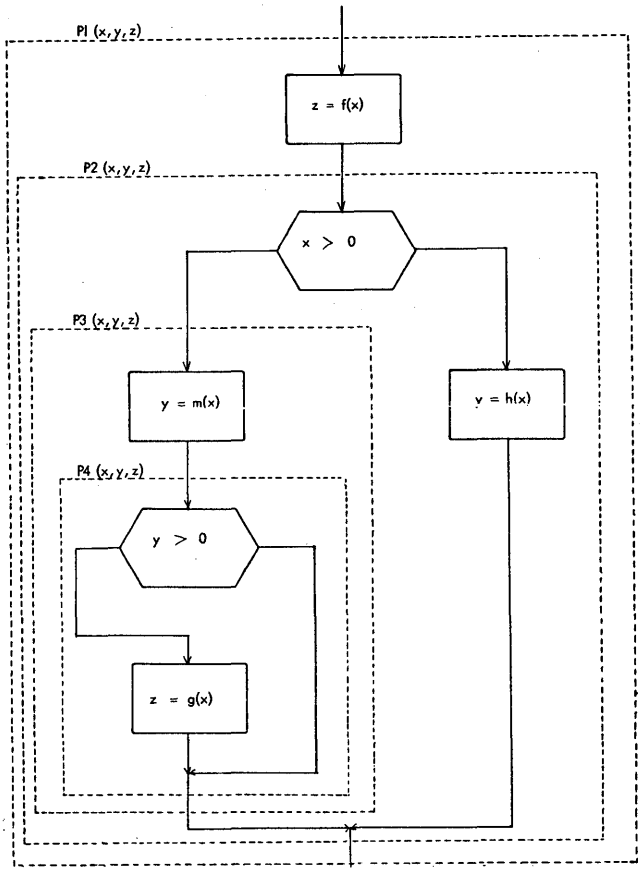


Figure 2—Structured version of program

The second example illustrates a basic result, namely that for HAC programs which consist of two prime programs with all variables being global, the minimum complexity occurs when the number of instructions is equally divided between the two subprograms. This result is obtained directly from a minimization of the expression for the complexity of such a program. Repeated applications of this result yield the conclusion that the minimal complexity for a program occurs when only a structured basis for the control is used, i.e., only the control structures sequence, if-then-else, do-while, repeat-until, do-while-do, if-then, and function.

This proposed measure has a demonstrated sensitivity to both control and data structuring within programs, and, because of its close relationship to the structured programming control graphs and prime programs, is felt to be a valuable tool in the quantification of overall complexity in the effort to formalize the intuitive concept of a good program.

REFERENCES

1. McCabe, T. J., A Complexity measure, *IEEE Trans. on Software Engineering* 2, No. 4, 1976, pp. 308-320.
2. Sullivan, J. E., Measuring the complexity of computer software, MITRE Corp., Report MTR-2648, June, 1973.
3. Hellerman, L., A measure of computational work, *IEEE Trans. on Comp.* 21, No. 5, 1972, pp. 439-446.
4. Savage, J. E., Computational work and time on finite machines, *Journal of the ACM* 19, No. 4, 1972, pp. 660-674.
5. Savage, J. E., *The complexity of computing,* J. Wiley and Sons, New York, 1976.
6. Chaitin, G. J., A theory of program size formally identical to information theory, *Journal of the ACM* 23, No. 3, 1975, pp. 329-340.
7. Linger, R. C. and H. D. Mills, *Structured Programming Theory and Practice,* Addison Wesley, Reading Mass., 1977 (to appear).
8. Parnas, D. L., A technique for software module specifications with examples, *Communications of the ACM* 15, No. 5, 1972, pp. 330-336.

A language for specifying software tests

by DAVID J. PANZL
General Electric Company
Schenectady, New York

INTRODUCTION

The execution of software test cases and the verification of test results may be performed automatically by a new type of program called an automatic software test driver. When using an automatic test driver, a formal test procedure is coded in a special test language. The test procedure takes the place of the test data and test setup instructions of conventional testing, and controls the automatic test driver. An automatic software test driver applies one test procedure to all or part of a target program, executes all of the test cases specified in the test procedure, and verifies that the results of each test case are correct. This paper describes the Fortran Test Procedure Language (TPL/F) which was developed at General Electric and is used for specifying test procedures for Fortran software.[1]

The concept of automatic software test drivers is a new idea that has been evolving slowly over the past six years and just now seems to be approaching the point where soon it may play a significant role in the development and maintenance of production software. Two other automatic software test drivers that were developed independently in recent years and have their own software test languages are described in References 2 and 3.

The Fortran Test Procedure system is illustrated in Figure 1. One test procedure coded in TPL/F and the source code for one or more modules of the target program are processed by the automatic test driver which executes all of the test cases specified in the test procedure, and produces a brief test execution report (Figures 2 and 3) stating which test cases failed, if any, and the degree of testing coverage actually achieved by the test procedure.

Test cases consist of input data for the target program and model output data. The automatic test driver actually executes the target program for each test case, feeding the input data to the target program and comparing outputs from the target program with the model outputs specified in the test procedure. Incorrect outputs produce a diagnostic in the test execution report (Figure 3).

Since the TPL/F system has access to the target program's source code, it can monitor which statements were actually executed and which branches were actually traversed while executing a test procedure. This type of information provides valuable feedback to the test designer and is generally not available when testing is done manually.

The specific goals of the TPL/F automatic software test driver are as follows. The need for writing drivers and stubs for module and subsystem testing is eliminated since the TPL/F system can test any combination of one or more modules independently of the rest of the target program. The TPL/F test language provides a standard format for specifying software tests and the test procedure processor provides a standard test execution setup. Since formal test procedures specify the correct outcomes of test cases, the test procedure processor automates the verification of test execution results.

TPL/F TEST PROCEDURES—AN OVERVIEW

TPL/F test procedures and the major elements of the TPL/F language are introduced in this section by means of an example. The following section describes the TPL/F language in greater detail. For the present example, we show a simple test procedure for the Fortran target module of Figure 4. The target module, IOSUB, is a subroutine and needs to be called by a higher level module for execution. IOSUB also invokes a lower level subroutine, LABEL, which is not under test and must be simulated when testing IOSUB. On each entry to IOSUB, one unformatted record is read from logical unit 12 and LABEL is called once.

A simple test procedure to test IOSUB is illustrated in Figure 5. The test procedure contains three test cases, each of which causes an actual execution of the target module, IOSUB. The MODULES statement names the target modules (IOSUB of Figure 4) under test.

TPL/F test cases are executed in three steps. First, initialization code is executed and puts the target program in a known initial state. In the example, the first two statements of each test case are used to initialize the target program. In the first test case, for example, they assign the values 1 and 0 to the variables N1 and N2, respectively, of IOSUB.

Second, the target program is executed. The form of the EXECUTE statement used in Figure 5 causes execution to begin at the first executable statement of IOSUB and terminate the first time a STOP or RETURN statement is encountered in IOSUB.

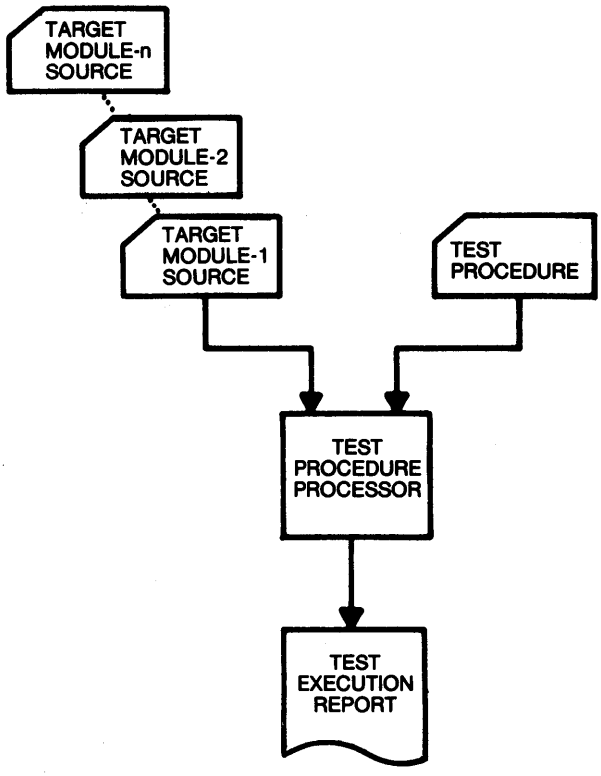


Figure 1—The TPL/F automatic software test driver

Third, after the test case execution has terminated, an assertion about the final state of IOSUB is verified. In the first test case of Figure 5, the variables N1 and N2 of IOSUB are verified to have the values of 6 and 0, respectively.

In TPL/F, stub versions of missing subprograms called by the target program are coded as Fortran-like subroutines and functions embedded in the test procedure. In Figure 5, for example, a stub version of subroutine LABEL begins at the SUBROUTINE statement and runs thru the following END statement. #TEST is an integer system variable whose value is always the index of the test case currently executing. The stub version of LABEL verifies that the argument passed by IOSUB when executing under the first test case is 10 and returns the value 0 for the argument on return. Should LABEL call IOSUB with an incorrect argument value, the ABORT statement would terminate the current test case (but not the test procedure) and put a diagnostic message in the test execution report.

The six statements beginning at the I/O SIMULATOR statement and continuing through the following END statement specify an I/O simulator for logical unit 12 of the target program. An I/O simulator can supply input data to the target program in response to READ statements or verify outputs from WRITE statements in the target program. The I/O simulator of Figure 5 specifies three unformatted records containing five integer values each, followed by an end-of-file mark. The first three records read by the target program from logical unit 12 will be taken from the I/O simulator. An attempt to read a fourth record would produce an EOF return in the target program.

```
ALL TEST CASES SUCCEEDED

STATEMENTS EXECUTED: 100%
BRANCHES TRAVERSED:  100%
```

Figure 2—A successful test procedure execution report

```
VERIFY FAILURE IN TEST CASE 2 AFTER TERMINATION
   (IOSUB:N1 .EQ. 0 .AND. IOSUB:N2 .EQ. 1)

STATEMENTS EXECUTED: 100%
BRANCHES TRAVERSED:  100%
```

Figure 3—A failed test procedure execution report

Although the test procedure in this example contains only three test cases, most test procedures contain a much larger number. Typically, a test procedure for a single Fortran target module of 50 to 100 statements may contain 20 to 50 test cases, depending on the complexity of the target module's control logic. Therefore, a more compact notation usually is required to represent test cases. TPL/F uses a built-in macro processor which allows the general form of a test case to be specified as a macro prototype and each specific test case is represented by a single macro call. In practice, test procedures tend to contain only a small number of forms of test cases (often only one) which are invoked many times with different specific data values. Figure 6 shows the three test cases of Figure 5 as they would appear when coded as TPL/F macro calls.

THE TPL/F TEST LANGUAGE

The previous section introduced the control structure and the four major components of TPL/F test procedures (test cases, subprograms, I/O simulators, and macros). This section contains an informal and intuitive introduction to all of the constructs of the TPL/F test language. Complete descriptions of the TPL/F language and the Fortran test procedure system are contained in References 4 and 5.

Figures 7 thru 13 describe the syntax of TPL/F by means of syntax graphs. Rounded boxes represent terminal symbols of the language. Rectangular boxes are non-terminal symbols requiring further definition. The following Fortran syntactic forms (marked with an asterisk in the syntax

```
SUBROUTINE IOSUB(N1,N2)
  .
  .
CALL LABEL(ICOUNT)
  .
  .
READ (12) (K(I),I=1,5)
  .
  .
END
```

Figure 4—A sample target module

```
MODULES IOSUB

I/O SIMULATOR 12
RECORD (1) 0, 5, 1, 2, 7
RECORD (2) 6, 1, 7, 3, 4
RECORD (3) 2, 9, 8, 0, 0
RECORD (4) #EOF
END

SUBROUTINE LABEL(I)
INTEGER IN(3),IOUT(3)
DATA IN/10,20,0/, IOUT/0,0,1/
IF (I .NE. IN(#TEST)) ABORT
I = IOUT(#TEST)
RETURN
END

IOSUB:N1 = 1
IOSUB:N2 = 0
VERIFY (IOSUB:N1 .EQ. 6 .AND. IOSUB:N2 .EQ. 0)
EXECUTE

IOSUB:N1 = 0
IOSUB:N2 = 100
VERIFY (IOSUB:N1 .EQ. 0 .AND. IOSUB:N2 .EQ. 1)
EXECUTE

IOSUB:N1 = 500
IOSUB:N2 = 501
VERIFY (IOSUB:N1 .EQ. 1 .AND. IOSUB:N2 .EQ. 1)
EXECUTE

FIN
```

Figure 5—IOSUB test procedure coded in the TPL/F test language

graphs) are embedded in TPL/F. Their grammer depends on the syntax of the host Fortran dialect.

identifier	declaration statement
label	format
integer constant	I/O list
string	data list
relational operator	logical expression
subprogram statement	

The symbol "Ftn statement☆" refers to any statement of the host Fortran dialect.

Test procedures

Test cases are the primary element of the control structure of a test procedure (Figure 7). Executing a test procedure consists of executing each of its test cases. Subprograms and I/O simulators defined in the test procedure only influence the performance of test cases. Fortran declaration statements may be included in test procedures to define COMMON blocks whose variables are referenced by the test procedure.

The MODULES statement names all of the target modules affected by the test procedure. The first name on the MODULES statement is the *primary target module*. All test procedure references to target program variables or statement labels, not qualified by an explicit target module name, are directed to the primary target module.

```
MACRO MAC(P1,P2,P3,P4)
IOSUB:N1 = P1
IOSUB:N2 = P2
VERIFY (IOSUB:N1 .EQ. P3 .AND. IOSUB:N2 .EQ. P4)
EXECUTE
MEND

MAC(1,0,6,0)
MAC(0,100,0,1)
MAC(500,501,1,1)
```

Figure 6—The three test cases of Figure 5 recorded using a TPL/F macro definition

Test cases

Each test case (Figure 8) in a test procedure causes an actual execution of the target program.

The EXECUTE statement specifies where to begin and where to terminate a test case execution, and how many times to execute it. For example, the following statement would cause its test case to be executed three times, each time beginning at statement label 50 of the primary target module and stopping whenever either statement label 25 of target module SUB2 or statement label 35 of target module SUB3 is executed

```
EXECUTE*3 FROM :50 TO SUB2:25,SUB3:35
```

An exclamation mark preceding a termination label reference means that the test case is to terminate without executing that statement. Otherwise, it would be terminated immediately afterward. The beginning statement, when specified, is always executed, regardless whether or not its reference is preceded by an exclamation mark. If no beginning label is specified, execution starts at the first executable statement of the primary target module. Regardless whether or not termination labels are specified, test cases always terminate whenever a STOP statement or a RETURN statement in the primary target module is executed.

All test case statements other than EXECUTE, VERIFY, and USE are executed immediately prior to the start of a test case execution. These are usually modified Fortran statements (see below) supplying initial values to the target program.

The VERIFY statement specifies an assertion about the target program. The AT clause, if present, specifies when, during the test case execution, the assertion is to be verified. The following statement causes an assertion to be verified whenever statement label 100 in SUB1 is executed, as well as immediately after the test case terminates.

```
VERIFY AT SUB1:100,#TERM (SUB1:N .EQ. 0)
```

An exclamation mark preceding an AT clause label reference causes the assertion to be verified immediately before executing the specified statement. Otherwise, verification occurs after executing the referenced statement. When no AT clause is specified, the assertion is verified after the test case terminates. An attempt to verify a false assertion ter-

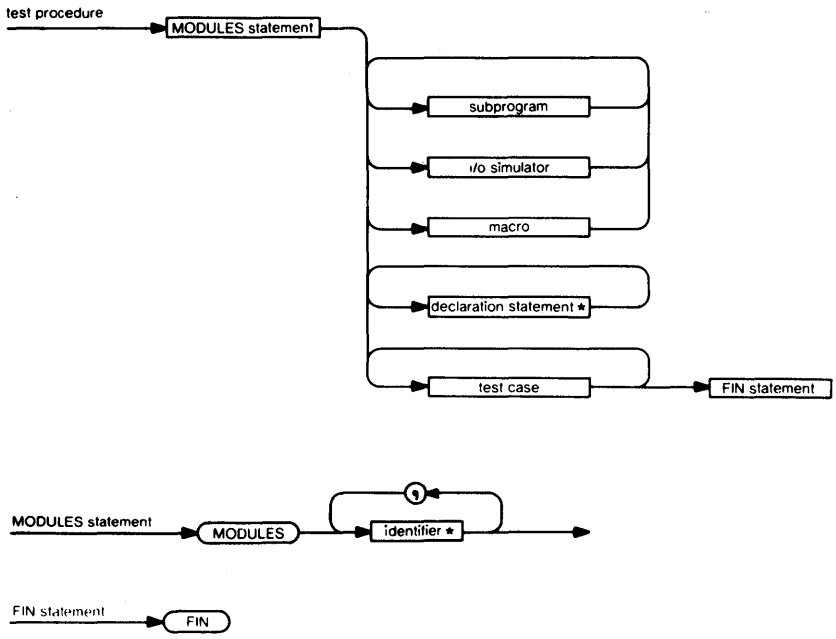


Figure 7—Test procedure syntax

minates the current test case (but not the test procedure) and causes an error message to appear in the test execution report as in Figure 3.

The USE statement modifies the mapping of target program I/O references to I/O simulators and affects only the test case in which it appears. For example, if a test procedure contains I/O simulators for logical units 20 and 30, all target program I/O references to these logical unit numbers would ordinarily invoke I/O simulators for those units in the test procedure. The following statements appearing in a particular test case, however, would direct all target program references to logical unit 20 to I/O simulator 30; and all target program references to logical unit 30 to an actual logical unit of that number.

```
USE 30 FOR 20
USE #ACTUAL FOR 30
```

The #SCRATCH clause directs target program I/O operations to a temporary scratch file.

Subprograms

Fortran subroutines and functions defined in a test procedure (Figure 9) may be referenced inside the test procedure or may be used as subprogram stubs.

When referenced inside a test procedure, subroutines may be used to perform complicated recurrent test case initialization tasks or Boolean functions may be used to verify complicated assertions about the target program. When used as subprogram stubs, subprograms defined in the test procedure are surrogates for missing subprograms called by the target program. Any subprogram defined in a test procedure can be used in either manner.

I/O simulators

I/O simulators are used to simulate I/O devices and files referenced by the target program. An I/O simulator (Figure 10) is a closed subroutine and is entered each time its logical unit number is referenced by a target program I/O statement.

Each I/O simulator has a unique record counter which always points to the current record on the simulated I/O unit. Prior to each test case execution, all record counters are reset to zero. While executing a test case, each time the target program reads or writes the simulated unit, its record counter is incremented.

I/O simulators can simulate either random or sequential devices. The NO RESET clause on the I/O SIMULATOR statement preserves the record counter value from one test case to another. The NO REWIND clause inhibits target program REWIND statements on the I/O simulator. Examples:

```
I/O SIMULATOR 12 RANDOM(1000)
I/O SIMULATOR 16 NO RESET
```

The first statement defines a random I/O simulator for logical unit 12 with a maximum record index of 1000. The second statement defines a sequential I/O simulator for logical unit 16 whose record counter is not reset between test case executions. The latter form is used for providing multiple test cases with different data files from the same logical unit.

The RECORD statement is the primary I/O simulator statement for simulating I/O devices and files. When the target program executes a READ or WRITE statement on a simulated logical unit, the target program is interrupted and control passes to the I/O simulator. The I/O simulator executes until an operable RECORD statement is encoun-

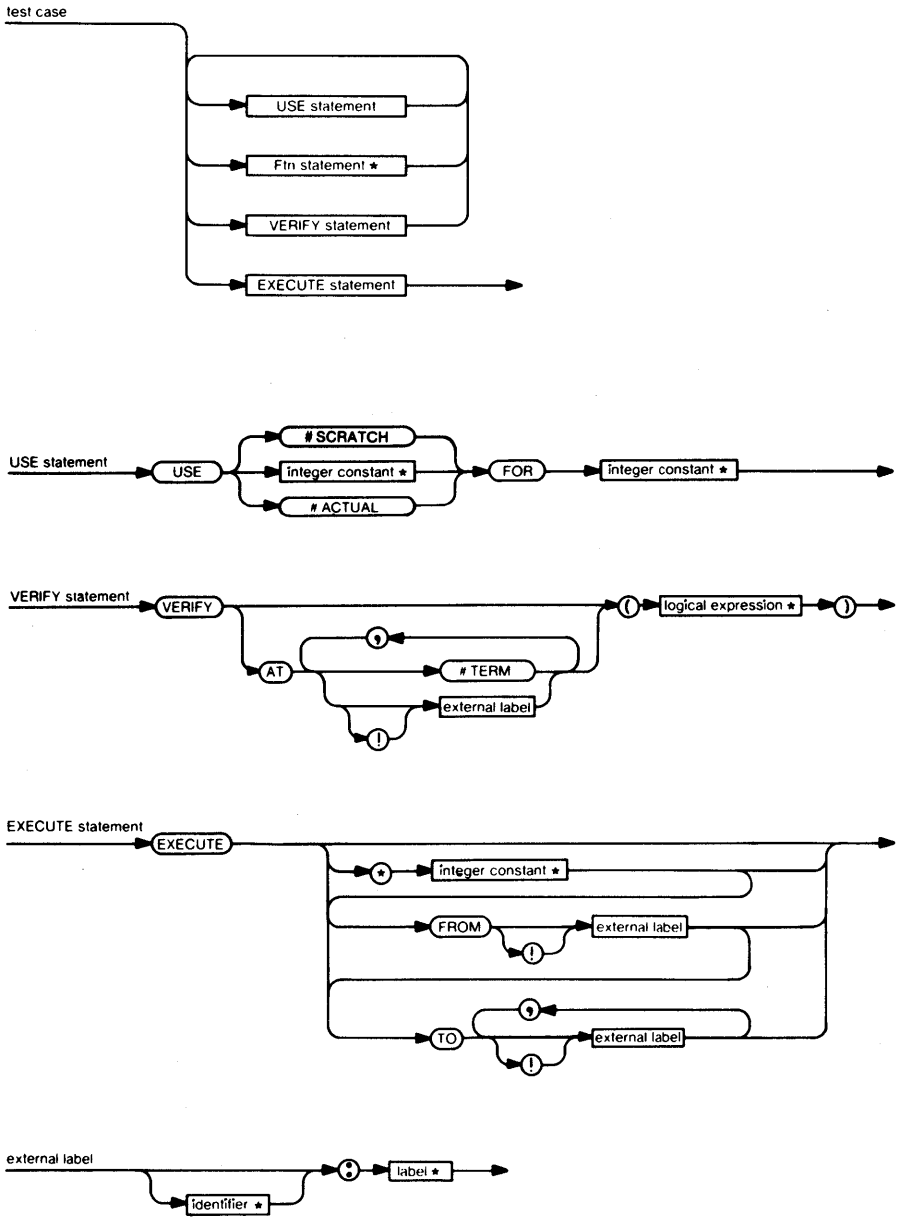


Figure 8—Test case syntax

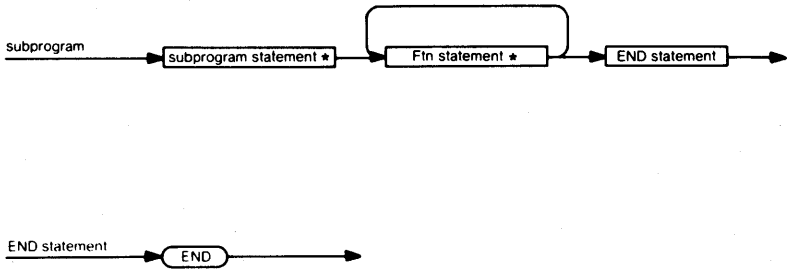


Figure 9—Subprogram syntax

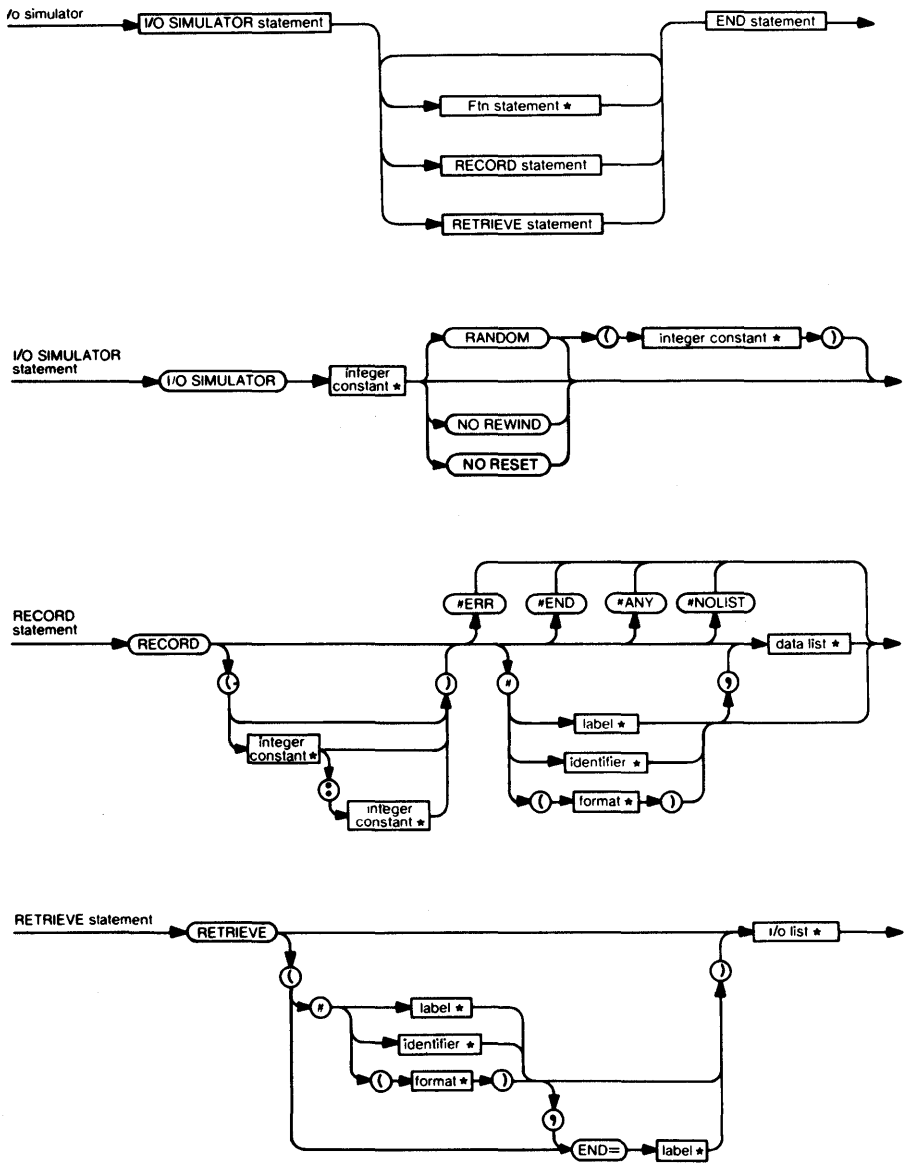


Figure 10—I/O simulator syntax

tered. If the I/O simulator was invoked by a target program READ statement, input data values from the RECORD statement are passed to the target program READ list and control is returned to the target program. If the I/O simulator was invoked by a target program WRITE statement, the data output by the WRITE statement is captured and compared with data in the RECORD statement. If the comparison is satisfied, control returns to the target program; otherwise an error message is written in the test execution report.

RECORD statements that specify a record range are operable only when the value of the host I/O simulator's record counter falls within the specified range. Otherwise, the RECORD statement is treated as a no-op. A record range may be specified as a single record number as in

RECORD (5) . . .

or as a minimum and maximum value as in

RECORD(5:10) . . .

The first statement is operable only for the fifth record on the simulated device and the second statement for the fifth thru tenth. A RECORD statement with no record range is always operable.

A RECORD statement may describe either formatted or unformatted records, depending on whether or not a format reference, indicated by a #, is specified, as in the following examples.

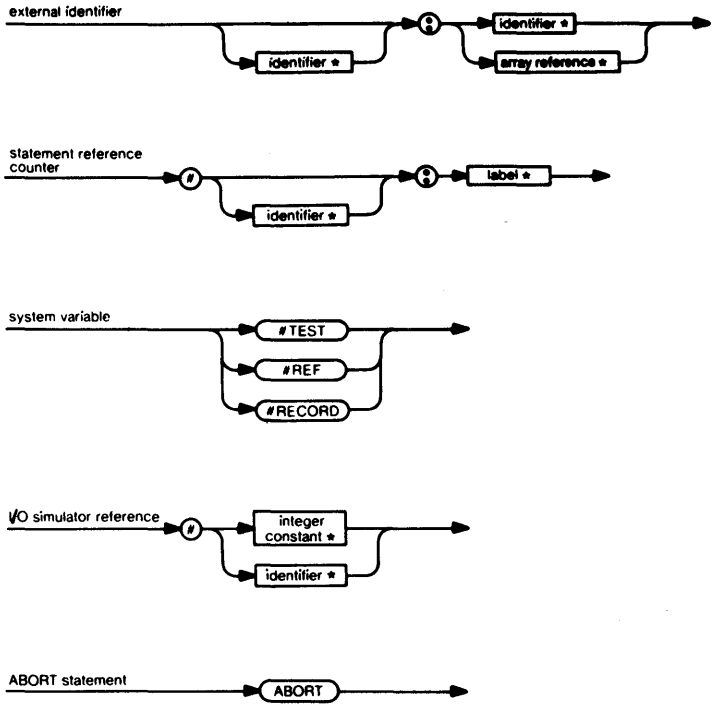


Figure 11—TPL/F modifications to embedded Fortran statements

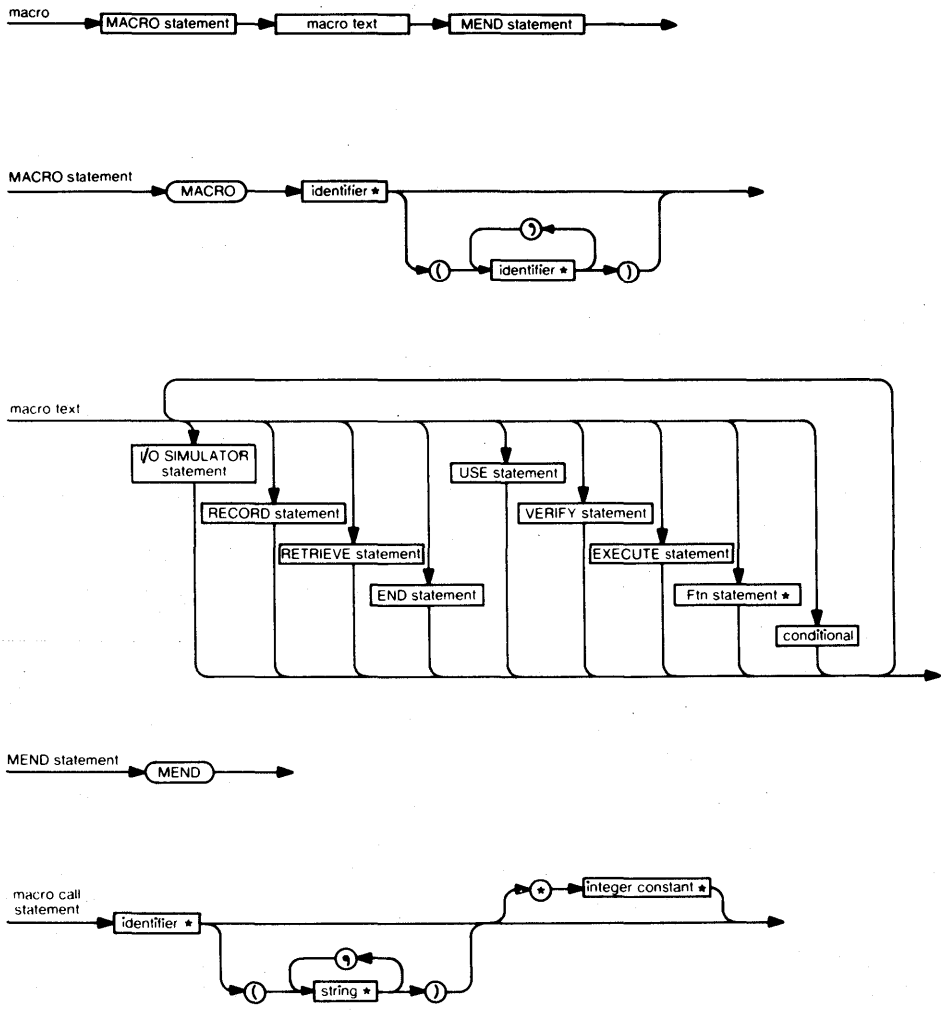


Figure 12—Macro syntax

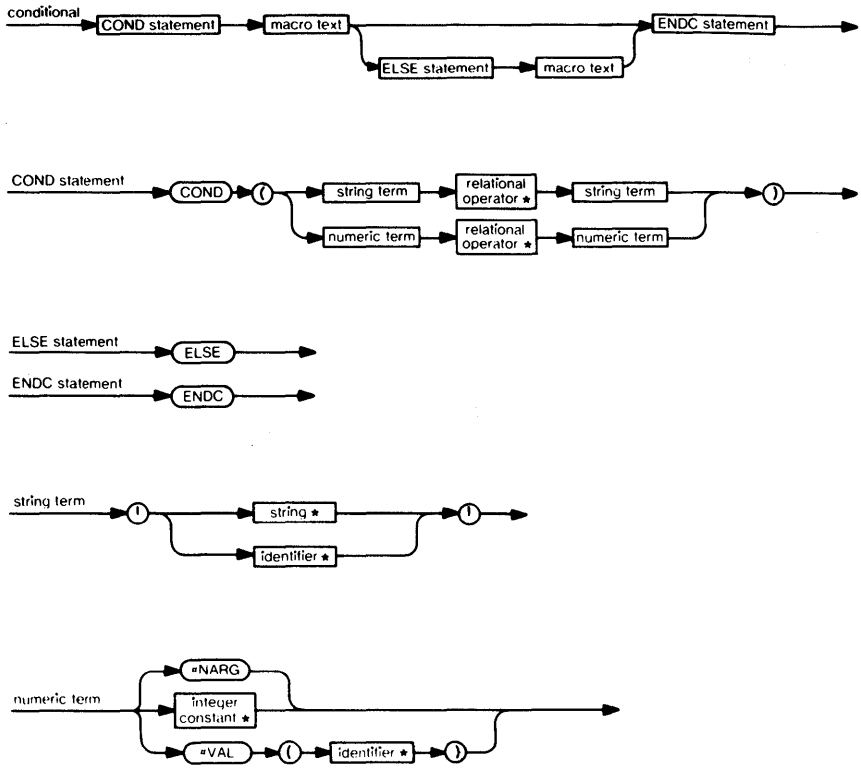


Figure 13—Conditional construct syntax

RECORD #(3I5), 10,20,30
RECORD #999, (X(I),I=1,10)
RECORD 7,12,(18,I=1,10)

The first two statements simulate formatted records and the third, an unformatted record. Formatted records must be generated in response to target program formatted I/O statements and unformatted records must be generated in response to target program unformatted I/O statements.

The #ANY clause in a RECORD statement verifies that the I/O simulator was invoked by any target program write operation. The #NOLIST clause verifies that the I/O simulator was invoked by a formatted write operation with no I/O list and is used primarily for skipping titles and formatting information when verifying formatted output records.

The #ERR clause in a RECORD statement simulates an I/O error and causes the ERR=label exit to be taken from the corresponding target program I/O statement. The #END clause simulates an end of file condition and is used to verify a target program ENDFILE statement or cause the END=label exit to be taken from a target program READ statement.

The RETRIEVE statement is used for verifying complex assertions about target program outputs. When the host I/O simulator is invoked by a WRITE or PRINT statement, the output record is captured, optionally interpreted according to a format, and assigned to variables in an I/O list. While the RECORD statement is limited to simple tests of equality between output variables and known values, the RETRIEVE statement makes output records available for any type of test. The first example of the following section (EXAMPLES) illustrates the use of the RETRIEVE statement.

TPL/F modifications to embedded Fortran statements

Fortran statements may appear throughout test procedures. The initialization part of a test case definition, for example, usually consists of Fortran code only. When appearing inside a test procedure, any Fortran statement may be modified as follows.

External identifiers, statement reference counters, and system variables (Figure 11) may be referenced wherever a variable would usually appear in a Fortran statement.

An external identifier refers to a variable or array in the target program. The initialization code and assertions in the example of the previous section contained external identifiers pointing to the variables N1 and N2 of target module IOSUB.

Statement reference counters provide a means for verifying the number of times a target program statement is executed. Every test procedure reference to a statement reference counter causes an integer-valued counter to be associated with the indicated statement in the target program. Prior to each test case execution, all statement reference counters are reset to zero. During test case executions, statement reference counters are incremented each time their associated statements are executed. The following

statement, for example, verifies that the statement at label 100 of target module SUB1 was executed exactly five times during the current test case execution.

```
VERIFY (#SUB1:100 .EQ. 5)
```

Three special symbols are used to reference system variables whose values are controlled automatically by the test procedure processor.

The value of #TEST is the index (e.g., first, second, etc.) of the currently executing test case.

#REF is defined as follows. Every subprogram and I/O simulator has a reference counter associated with it. All reference counters are reset to zero prior to each test case execution and incremented by one on each entry to their associated subprograms. Inside a subprogram or I/O simulator, #REF has the value of the subprogram or I/O simulator's reference counter. In all other places, #REF is always equal to zero.

Inside an I/O simulator routine, #RECORD has the value of the I/O simulator's record counter. Elsewhere, the value of #RECORD is always zero. Examples:

```
Y=VAL(#REF,#TEST)
IF (#RECORD .EQ. N) GO TO 100
```

Ordinarily, Fortran I/O statements embedded in a test procedure are not affected by I/O simulators. A logical unit designator prefixed by a pound sign in an embedded Fortran I/O statement temporarily suspends this rule and causes the I/O statement to be treated the same as a target program reference to the logical unit. For example, the following statement appearing inside a test procedure references the I/O simulator for logical unit 10, not the actual logical unit 10.

```
READ (#10) X,Y,Z
```

The ABORT statement is used inside subprograms and I/O simulators to terminate the current test case when an error is detected. Terminal assertions of aborted test cases are not verified.

Macros

The primary role of macro definitions (Figure 12) is that of an efficient shorthand notation for describing test cases. Macro definitions are entered in a macro definition library under the symbolic name specified on the macro statement. When invoked by a reference to the macro name, formal arguments appearing in the text of the macro definition are replaced by actual arguments supplied by the call. The modified text of the macro definition then replaces the macro call statement.

The conditional construct (Figure 13) allows statements to be conditionally included in a macro expansion. The Boolean expression in the COND statement is evaluated each time its host macro is expanded. If the expression evaluates to true, all statements following the COND statement, down to the ELSE statement, if present, are included in the macro expansion. When the COND statement expression evaluates to false, only those statements between the ELSE statement, if included, and the ENDC statement are included in the macro expansion.

```
MACRO MAC(N,ALPH)
COND (#NARG .EQ. 1)
:A = 0.
  COND(#VAL(N) .NE. 0)
  :B = 0.
  ENDC
ELSE
:C = 0.
  COND (ALPH .EQ. 'ABC')
  :D = 0.
  ENDC
ENDC
MEND
```

Macro Call	*Macro Expansion*
MAC(0)	:A = 0.
MAC(2)	:A = 0.
	:B = 0.
MAC(2,XYZ)	:C = 0.
MAC(2,ABC)	:C = 0.
	:D = 0.

Figure 14—Conditional macro expansion examples

String terms in COND statement expressions are either quoted strings or formal arguments of the host macro definition which are replaced at macro expansion time by strings.

Two special forms of numeric terms may appear in COND statement expressions. #NARG is the number of actual arguments supplied in the current call on the host macro. The numeric value of #VAL(arg) is determined as follows. Usually, arg is a formal argument in the host macro definition. After argument substitution, if arg is a numeric constant, then that is the numeric value of #VAL(arg); otherwise the macro expansion terminates in error. Figure 14 illustrates conditional macro expansions.

EXAMPLES

The test procedure example in the beginning of this paper illustrates the basic structure of test case definitions, the use of macro definitions for replicating test cases, I/O simulators and subprogram stubs. Two additional examples in this section illustrate common styles used for writing test procedures.

Figures 15 and 16 illustrate the use of I/O simulators for testing I/O-oriented target modules. The target module of

```
  1 READ (12,100,END=300) X,Y
      .
200   .
      .
300 WRITE (6,150) W
    STOP
    END
```

Figure 15—An I/O-oriented target module

```
      MODULES MAIN
C
C  INPUT VALUES
C
      I/O SIMULATOR 12 NO REWIND
      RECORD (1) #1, 0.0,0.0
      RECORD (2) #END
      RECORD (3) #1, 0.01,0.00001
      RECORD (4) #1, 0.01,0.00002
      RECORD (5) #END
      RECORD (6) #1, 1.0,5.0
      RECORD (7) #, 1, 2.0,4.0
      RECORD (8) #1, 3.0,3.0
      RECORD (9) #END
      RECORD (10) #1, 10000.0,0.00001
      RECORD (11) #END
    1 FORMAT('BASE',F12.6,10X,'VALUE',F12.6)
      END
C
C  OUTPUT VALUES
C
      I/O SIMULATOR 6 NO REWIND
      DIMENSION X(4)
      DATA X/0.0, .0.0132,2.5,1.0/
      RETRIEVE (#1) VAL
      IF (ABS(VAL-X(#TEST)) .GT. 0.00005) ABORT
      RETURN
    1 FORMAT('VALUE =',F14.8)
      END
C
C  DEFINE GENERAL TEST FOR MAIN
C
      MACRO TSTPRG(ZP)
      VERIFY AT :200 (:ABS(Z-ZP) .LT. 0.0005)
      EXECUTE
      MEND
C
C  FOUR TEST CASES
C
      TSTPRG(1.0)
      TSTPRG(1.001)
      TSTPRG(5.0)
      TSTPRG(100.0)
      FIN
```

Figure 16—A test procedure for the target module of Figure 15

Figure 15 is a main program that reads formatted records from logical unit 12 until an end-of-file condition is encountered. After some processing, the program writes one formatted record on logical unit 6 and terminates.

The test procedure in Figure 16 exercises the single target module, MAIN. Four test cases each supply inputs to MAIN on logical unit 12 and verify its outputs on logical unit 6. Additionally, a non-terminal assertion is verified each time control reaches MAIN:200.

Figures 17 and 18 contains another example illustrating

```
SUBROUTINE SUB2(M)
DIMENSION M(8,3)
  .
  .
  .
END
```

Figure 17—A target module

```
      MODULES SUB2
C
C  INPUT VALUES
C
      I/O SIMULATOR 10 NO REWIND
      RECORD (1:3) 0,0,0,0,0,0,0,0
      RECORD (4:6) 1,1,1,1,1,1,1,1
      RECORD (7:9) 1,0,0,0,0,0,0,1
      RECORD (10:12) 0,1,1,1,1,1,1,0
      END
C
C  OUTPUT VALUES
C
      I/O SIMULATOR 20 NO REWIND
      RECORD (1) 1,0,4,8,0,1,5,0
      RECORD (2) 1,1,0,1,5,3,6,0
      RECORD (3) 2,0,1,1,8,1,6,4
      RECORD (4) 7,9,1,6,4,6,6,9
      RECORD (5) 1,7,9,1,4,1,9,4
      RECORD (6) 6,2,5,9,0,3,6,2
      RECORD (7) 0,7,2,0,9,6,9,9
      RECORD (8) 9,5,7,0,9,1,2,9
      RECORD (9) 1,9,0,7,0,0,2,2
      RECORD (10) 3,6,8,4,6,5,7,3
      RECORD (11) 2,5,5,9,5,8,5,3
      RECORD (12) 9,3,3,0,9,9,5,8
      END
C
C  VERIFY OUTPUTS
C
      FUNCTION OUTVFY
      WRITE (#20) ((:M(I,J),J=1,8),I=1,3)
      RETURN
      END
C
C  DEFINE GENERAL TEST FOR SUB2
C
      MACRO TSTSB2
      READ (#10) ((:M(I,J),J=1,8),I=1,3)
      VERIFY (OUTVFY)
      EXECUTE
      MEND
C
C  FOUR TEST CASES
C
      TSTSB2*4
      FIN
```

Figure 18—A test procedure for the target module of Figure 17

the use of internal references to I/O simulators for generating large amounts of data for test case initialization and verification. The target module, SUB2, of Figure 17 performs a transformation on the array M.

The test procedure of Figure 18 contains four test cases, each of which supplies SUB2 with 24 input values for the array M and verifies the contents of M returned by SUB2. The input and output values are contained in two I/O simulators read only by the test procedure.

DISCUSSION

The notation that software tests can be specified in a formal language, just as are computer programs, will likely improve the quality of software test design since, for the first time, software tests reside in a medium (the test pro-

cedure) that is both executable and readable and therefore available for peer review and criticism. Automatic software test drivers should also influence the quality of software test design by providing feedback on the degree of testing thoroughness actually achieved by a test procedure.

Perhaps the strongest effect of automatic software test drivers and formal test procedures will be on the maintenance of production software. It is widely agreed that the majority of errors appearing in production software are introduced as unintended side effects of post-release program modifications. This is due, in large part, to the absence of a convenient mechanism for preserving test cases throughout a program's life-cycle so that they can be used to check out post-release program modifications. In order to effectively retain software tests for post-release regression testing, it is necessary that someone other than the original test designer be able to execute them and verify the correctness of test results. Formal test procedures should facilitate the retention of software test cases throughout the entire life cycle of computer programs since they are completely self-contained, they execute automatically without special test set-up instructions, and they are self-checking. The adoption of automatic software test drivers should therefore lead to significant software cost savings since it is often estimated that up to 70 percent of the life-cycle costs of computer software are attributable to testing and debugging.

REFERENCES

1. Panzl, D. J., "Test Procedures—A New Approach to Software Verification," in *Proceedings Second International Conference on Software Engineering*, October 1976.
2. "Automated Unit Test (AUT) Program Description/Operation Manual," IBM Installed User Program Number 5796-PEC, August 1975.
3. "Testmanager: Training Manual," TMR-008, Management Systems and Programming, Ltd., 71 Gloucester Place, London W1H 3PF, England, March 1976.
4. "Fortran Test Procedure Language—Programmer Reference Manual," General Electric Co., Schenectady, New York 12345, 1977.
5. "Test Procedure Processor—User Guide," General Electric Co., Schenectady, New York 12345, 1977.

A software quality plan for higher education—An abstract

by BARRY L. BATEMAN and CHADWICK H. NESTMAN
Southern Illinois University
Carbondale, Illinois

As the number of institutions of postsecondary education facing financial exigency increase, a method of tracking the progress of software projects in a systematic and unbiased manner is extremely critical. The software quality plan commonly found in business and industry can aid the college and university in developing guidelines and tests for the software development process, thus reducing costs.

Traditionally, testing and evaluation of software has been a minor or non-existent part of the programming effort within a university. If testing and evaluating does exist, it has normally been carried out by the programmer responsible for project development or by his close associate. This could mean that objectivity and selectivity in testing and evaluating can be lost because of a programmer's proximity to his work. This could also mean that more of the operating budget of the computing center has to be spent on testing and debugging after installation has been carried out. Testing and debugging in the user's environment can cause unnecessary problems in public relations and system acceptance. Some industrial companies, such as TRW, Inc. and the NCR Corporation, have recognized this problem and have established formal Software Quality Assurance (SQA) departments or interest groups whose responsibilities include the task of assuring that software is free from as many errors as possible. In carrying out their functions, the SQA organization utilizes a formal plan or guideline known as the Software Quality Plan (SQP). The SQP is a document prepared by each of the participating internal departments within computer services (programming, operation, training, production, etc.) and the customer or client. The plan, once prepared, spans the entire development cycle and becomes a coordinating agent between all interested groups in the establishment of activities, schedules, commitments and budgets.

Within the institution of postsecondary education, responsibility for the development of the SQA should be under the direct supervision of the Director of Software Quality Assurance or his designate. In institutions where there is no formal quality assurance office, the plan could be produced by the internal auditor or at least by the chief officer in charge of computing services. Although the SQP is initiated early in the development cycle, it must be updated as the project moves from one development phase to another. When writing a SQP, the following criteria must be considered: financial and budgetary analysis, manpower requirements (by departments), program dependencies (hardware and software), milestone schedules, and commitments by the intended user in relation to their support and acceptance criteria. Test plans and responsibilities should clearly be outlined including the control procedure, methodology, limitations, and acceptance criteria for each test to be used. It is suggested that there be at least a review or test during the feasibility stage, in order to check architectual proposals; the coding stage, in order to assure proper design; pre-installation, in order to exercise programs with near real data; installation to insure users are adequately trained and that proper documentation is available. Resource requirements, in the sense of stating needed hardware, software, special test tools, additional manpower and required documentation should also be provided in the SQP.

In essence, software quality within the university framework is a necessity, especially in light of financial exigency. This quality can be attained by better control during the project development process and by assuring test objectivity by having the evaluation of software carried out by non-partisan people—people not involved with the actual writing and coding of the software. These non-partisan persons would be part of a Software Quality Department and would be responsible for the implementation of a Software Quality Plan, that is, a guideline for the monitoring of software quality throughout the development process. Above all else the plan would describe the various tests to be conducted, responsibilities in carrying out those tests, and the results expected. This plan is not intended as a panacea for all postsecondary quality ills, but is intended as a starting point for the assuring of quality software in the university environment by reducing costs normally incurred when errors are corrected on the "fly."

The design of a prototype mutation system for program testing*

by TIMOTHY A. BUDD, RICHARD J. LIPTON and FREDERICK G. SAYWARD
Yale University
New Haven, Connecticut

and

RICHARD A. DEMILLO
Georgia Institute of Technology
Atlanta, Georgia

INTRODUCTION

When testing software the major question which must always be addressed is "If a program is correct for a finite number of test cases, can we assume it is correct in general." Test data which possess this property is called Adequate test data, and, although adequate test data cannot in general be derived algorithmically,[1] several methods have recently emerged which allow one to gain confidence in one's test data's adequacy.

Program mutation is a radically new approach to determining test data adequacy which holds promise of being a major breakthrough in the field of software testing. The concepts and philosophy of program mutation have been given elsewhere,[2] the following will merely present a brief introduction to the ideas underlying the system.

Unlike previous work, program mutation assumes that competent programmers will produce programs which, if they are not correct, are "almost" correct. That is, if a program is not correct it is a "mutant"—it differs from a correct program by simple errors. Assuming this natural premise, a program P which is correct on test data T is subjected to a series of mutant operators to produce mutant programs which differ from P in very simple ways. The mutants are then executed on T. If all mutants give incorrect results then it is very likely that P is correct (i.e., T is adequate). On the other hand, if some mutants are correct on T then either: (1) the mutants are equivalent to P, or (2) the test data T is inadequate. In the latter case, T must be augmented by examining the non-equivalent mutants which are correct on T: a procedure which forces close examination of P with respect to the mutants.

At first glance it would appear that if T is determined adequate by mutation analysis, then P might still contain some complex errors which are not explicitly mutants of P. To this end there is a COUPLING EFFECT which states that test data on which all simple mutants fail is so sensitive that it is highly likely that all complex mutants must also fail.

Readers wishing a further exposition of the ideas of mutation and substantiation of the assumptions made are referred to References 2 and 10.

THE SYSTEM

A pilot system has been built to implement mutation analysis on programs written in a subset of FORTRAN. The key features of this system are summarized in Figure 1. The system itself consists of 10,000 lines of FORTRAN code, and required six man months to design, implement and debug.

Notice we claim the system is man/machine interactive. In general an attempt is made to assign tasks to both the user and the machine processors which are best suited to using their particular capabilities. One way to see this is to view the system as a sort of "Devils Advocate", which when confronted with a program asks very difficult questions about the motivation behind it ("why did you use this type of statement here, when an alternative statement works just as well?"). The job of the human is then to provide justification (in the form of test data), which will give an answer to such questions.

An overview of the structure of the system is given in Figure 2. We point out that the language FORTRAN was chosen for the first implementation merely as a matter of convenience since it is in common use and there is a large body of software in existence to experiment on. The heart of the system (roughly that shown within the dotted box) is however, language independent, and given a sufficiently general internal form to implement a new language one would merely write a new input/output interface. Projects are currently under way to implement mutation analysis on

* This research was supported in part by NSF Grant MCS76-81486 and U.S. Army Research Grant DAAG-29-76-G-0338.

INTERACTIVE
MACHINE INDEPENDENT
LANGUAGE INDEPENDENT STRUCTURE
MODULAR DESIGN
INTENSIVE MAN/MACHINE INTERACTION

Figure 1—Key features of the pilot mutation system

COBOL and C (an ALGOL like language) using the structure represented by the box contained in the dotted lines.

An attempt was also made to keep the structure of the system largely machine independent. The system was originally programmed to run on a PDP-10 at Yale University. Currently we are in the process of transferring it to a CDC 7600 at the Georgia Institute of Technology.

A single run of a mutation system divides naturally into three phases the RUN PREPARATION phase, in which the necessary variables to send to the mutation executor are defined, the MUTATION phase, in which the actual mutations are produced and executed, and the POST RUN phase, in which results are analyzed and reports are generated. In the following we will describe in more detail the structure and effects of each phase.

The role of the run preparation phase is to initialize the various files and data buffer areas used by the mutation executor. It is characterized by a very interactive nature. The first object the user is requested to supply is the name of the file on which the FORTRAN subroutine resides. Then depending on whether PIMS has been run previously on this routine (in which case the internal form is stored on one of the many files PIMS constructs, see below) the subroutine is parsed into a concise internal format which is subsequently interpreted to simulate execution of the program. A

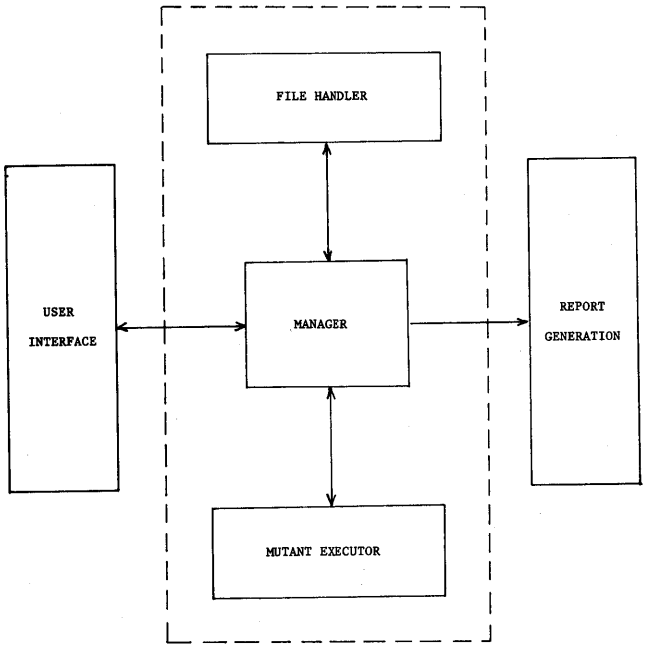


Figure 2

```
IF (A .LT. X(2)) P = 1

[SCALAR. A]
[ARRAY1. X]
[CONSTANT.2]
[AOP.SUBSCRIPT]
[ROP.LT]
[TRF.O]
[SCALAR. p]
[CONSTANT.1]
[ASSIGN.O]
```

Figure 3

fragment of the internal code generated for a given statement is shown in Figure 3.

The user is then interactively prompted for the test data on which the program and mutants are to be tested. After each test case has been specified the original program is executed on the test case and the results displayed so that the user may satisfy himself that the results produced are indeed correct.

After the test data has been entered the user is prompted for a listing of which mutant operators he wishes to enable. At present there are 25 mutant operators. These range from very simple low level ones, such as replacing each data occurrence (where a data occurrence is a scalar, constant or array reference) with all other syntactically correct data occurrences, to very high level mutations, such as deleting statements or altering the control structure of the program. A more detailed description of the mutations performed can be found in Reference 3.

Instead of constructing multiple copies of the program, for each mutant a short (four word) description of the mutation to be performed is kept. Each time the mutant is to be run the original program is then mutated according to the contents of this descriptor.

After the user has specified to the system his program, test data and the mutant operators he wishes applied, the system then enters the MUTATION phase. During this phase there is no user interaction. Mutation descriptor records are read in, one by one, and the mutation is produced. The mutant program is then executed on the test data and marked either "dead," meaning it produced results differing from the original program on at least one test case, or "living." A dynamic record is kept of the number and percentage of living mutants of each mutation type.

When all the mutant programs have been tested the post run phase is entered. In this phase statistics are displayed indicating the results of the mutation run. In addition the user can interactively view descriptions of those mutations which have survived. He can also specify that certain re-

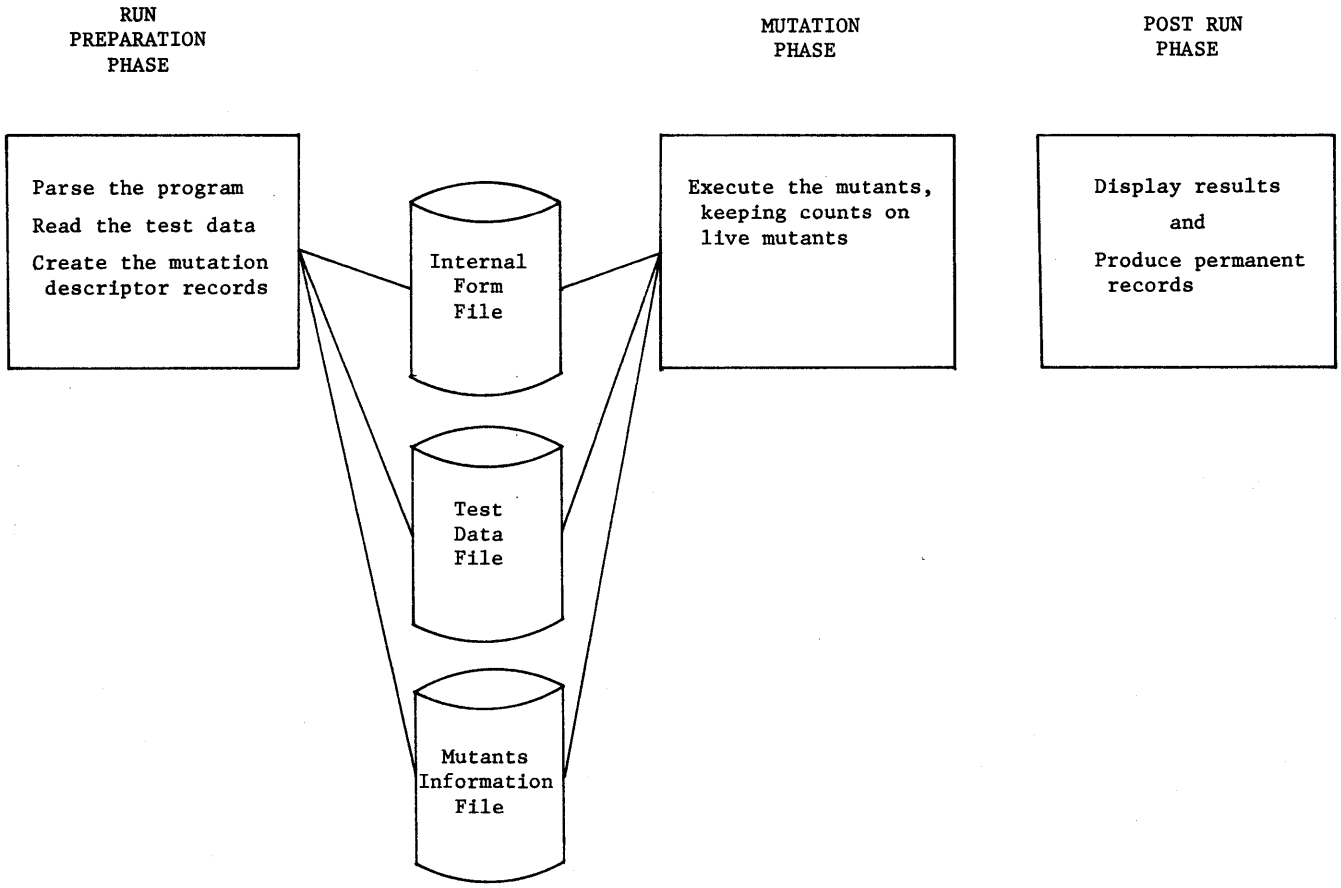


Figure 4

ports be generated in order to provide a detailed permanent record of the mutation run.

At this point, or at a later date, the user can re-run the system and augment his test data in an attempt to make the remaining mutants fail. He may also specify that additional mutant operators be applied to the program. This cycle can continue until the user is satisfied that the current test data adequately tests his program.

There are several files the system produces in order to store information from one run to the next. These are shown in Figure 4, which outlines the major functions of each phase. The internal form file stores the parsed version of the program. The test data file stores for each test case the test data input and the results of execution of that test data. The mutants information file keeps the mutant descriptor records plus various other counts on what types of mutants have been produced.

A COMPARISON OF PIMS TO OTHER DATA TESTING SYSTEMS

Various systems have been discussed in the literature for increasing confidence in the adequacy of test data, as the PIMS system does, or automatically constructing test data which meets some criterion. In this section we will report on experiments which show that the PIMS system is an improvement in this area over other systems which have been proposed.

The most widely known method of constructing test data automatically are those systems which utilize path analysis.[4-7] Essentially, these procedures attempt to construct data which force each statement to be executed at least once, and furthermore which transverse each feasible flow path through the code at least once. In some cases, such as loops, only an approximation to this can be made as the number of flow paths may be infinite. Here it is usual to just construct data which cause the loop to be executed at least twice.

These same objectives are met with mutant analysis in a number of ways, some directly by mutant operators, others indirectly by the coupling effect. There are mutants which cause each statement in the original program to be replaced by a TRAP statement, a special type of statement which if ever executed causes the program to immediately abort. Obviously, then if there is some statement in the program which is never executed, changing that statement to a TRAP statement will not alter the output of the program and hence will easily be detected.

Checking that every decision path is taken is essentially

```
      DO 10 I=1.J
           :
           :
      10 CONTINUE
=================
      DO 10 I=1.1
           :
           :
      10 CONTINUE
```

A LIVE MUTATION IF THE LOOP IS ALWAYS EXECUTED ONLY ONCE.

Figure 5

the same as checking that every predicate in the program evaluates at least once to both true and false. If this is not the case, say the predicate always evaluated to TRUE, then we can mutate the predicate in any way we desire as long as it retains this property of always remaining TRUE. These types of mutations are also usually quite obvious and easily detectable.

Mutation analysis can also insure that each loop is traversed at least twice. The only way a loop can be traversed only once (and all loops must be traversed at least once to pass the TRAP statement mutations) is if the terminating condition is the same as the starting condition. But in this case the mutant which replaces the terminating condition by the starting condition will survive (see Figure 5). This is once more easily detected.

With this, mutant analysis possesses all the capabilities of

```
      SUBROUTINE BSERCH(X.Y.N.A.IHIGH.LOW.ERR)
      INTEGER X(N).Y(N).N.A.IHIGH.LOW.ERR.MID
C     BINARY SEARCH PROCEDURE. IF X CONTAINS A ON RETURN
C     X(IHIGH) = A.IHIGH=LOW. IF NOT X(LOW) < A < X(IHIGH).
C     IF A IS OUT OF RANGE ON RETURN ERR CONTAINS 1
      ERR = 0
      IF ((X(1)-A).GT.0) GOTO 11
      IF ((A-X(N).LE.0) GOTO 5
 11   ERR = 1
      RETURN
 5    LOW = 1
      IHIGH = N
 6    IF ((IHIGH-LOW-1).NE.0) GOTO 7
      RETURN
 7    MID = (LOW+IHIGH)/2
      IF ((A-X(MID)).GT.0) GOTO 10
      IHIGH = MID
      GOTO 6
 10   LOW = MID
      GOTO 6
      END
```

Figure 6

```
Replace  IF ((IHIGH-LOW-1).NE.0) GOTO 7
by       IF ((IHIGH-LOW-1).GT.0) GOTO 7

Replace  MID = (LOW+IHIGH)/2
by       MID = (LOW+IHIGH)-2
```

Figure 7

path analysis systems which have been discussed in the literature.

Another class of systems for which extensive claims have been made are those which detect uninitialized variables and dead code.[8] Uninitialized variables are caught as a consequence of the interpretation process in the mutant system. Dead code is easily caught since an assignment made to a dead variable can be mutated in any way whatsoever and the program will remain the same.

A third class of systems for which there has recently been much discussion involves symbolic execution of the program. In one study[9] Howden analyzed 12 programs containing a total of 22 errors. He found that symbolic execution would catch 13 of those errors, while path analysis would discover only nine. In a similar study we estimated that mutation analysis, using only the mutant operators in the present PIMS system, would uncover 18 of the 22 errors. Of the remaining four, three would probably be discovered if we added two new mutant operators which the authors simply had not thought about. Hence, mutation analysis is in certain cases an improvement over symbolic execution.

As an example of the very subtle errors which mutation analysis can discover consider the program to perform binary search shown in Figure 6. If it happens that N=1 when the subroutine is called (i.e., the vector to be searched contains only a single element) then it is not difficult to see that the program will loop indefinitely. It is not clear that either symbolic execution or path testing would be sufficient to discover this error.

When mutant analysis is applied to this program there are two mutants generated (shown in Figure 7) which can only be eliminated by a test case consisting of one element. Hence the error is easily detected using mutant analysis. (There is a second error in this program which is also uncovered by mutant analysis. The discovery of that second error is left to the reader).

FUTURE WORK

There are several directions in which work is currently being pursued with respect to mutation analysis and the pilot mutation system. The most obvious is to show how a similar system might be built around another language, and research is under way to construct systems for COBOL and for C.

Another area of study is the design of an easy to use language for the description of test cases which allows for a variety of features. Test datasets can often be quite lengthy, yet two test cases can be very similar. Also, a user often wishes just to construct a number of random test cases following some specification. (Some of the pitfalls of using random data to test programs are discussed in Reference 10

where it is seen that mutation analysis can help in deriving "good" random test data.) Finding an easy yet powerful method of solving this problem is the goal of one area of research.

Finding a method to detect equivalent mutants is another area currently being pursued. It is often the case that a mutation will not produce a significantly different program (replacing the sequence I=1 J=1 with the sequence I=1 J=I is a trivial example). We have observed that programs tested have between one and two percent equivalent mutants. A method to automatically detect and remove equivalent mutants would allow us to provide even more significant measures of the adequacy of a test data set.

We point out that as a consequence of the modular design of the pilot system either of the above two major extensions can be added without a significant reprogramming effort.

A final area of current interest is the study of mutant operators. Certain operators seem to have a much greater ability to detect errors then others. Analysis of data along these lines would allow us to discover an order of application of mutant operators which would maximize the cost/benefit ratio.

CONCLUSIONS

It has been shown that the ideas of program mutation can be quickly and easily implemented as an interactive system for program testing. The resulting system represents a cost effective engineering approach to testing real world software. Large subroutines (over a hundred statements long) have been analyzed by our system with relative ease.

Mutation is a method of program testing which will significantly raise the level of reliability in both new and existing software, and is a major advance in the area of software testing.

REFERENCES

1. Goodenough, J. B. and S. L. Gerhart, "Towards a Theory of Test Data Selection," *IEEE Tran. Soft. Eng.*, SE-1,2, June 1975, pp. 156-173.
2. DeMillo, R., R. J. Lipton and F. Sayward, "PROGRAM MUTATION—A Method of Determining Test Data Adequacy," in preparation.
3. Budd, T. and F. Sayward, "Users guide to the Pilot Mutation System," Yale University Tech. Rep 114, 1977.
4. Ramamoorthy, C. V., S. F. Ho, and W. T. Chen, "On the Automated Generation of Program Test Data," *IEEE Trans. on Soft. Eng.*, SE-2,4, Dec. 1976, pp. 293-300.
5. Howden, W. E., "Methodology for the Generation of Program Test Data," *IEEE Trans. on Comp.*, C-24,5, May 1975, pp. 554-560.
6. Huang, J. C., "An Approach to Program Testing," *Computing Surveys*, 7,3, Sept. 1975, pp. 113-128.
7. Miller, E. F. and R. A. Melton, "Automated Generation of Testcase Datasets," Proc. 1st Int. Conf. on Reliable Software, *SIGPLAN Notices*, 10,6, June 1975, pp. 51-58.
8. Osterweil, L. J. and L. D. Fosdick, "Some Experience with DAVE—A Fortran Program Analyzer," *AFIPS Conference Proceedings*, Vol. 45, 1976, pp. 909-915.
9. Howden, W. E., "Symbolic Testing and the DISSECT Symbolic Evaluation System," *IEEE Trans. on Soft. Eng.*, SE-3,4, pp. 266-278.
10. DeMillo, R., J. Lipton and F. Sayward, "Hints on Test Data Selection," to appear in *Computer*, April 1978.

A panel session—User experience with new software methods

SESSION CHAIRMAN—VICTOR R. BASILI
University of Maryland

Panel Members

Donald J. Reifer—TRW
Donn Combelic—ITT
J. A. Rader—Hughes Aircraft Company
C. M. Bernstein—Exxon Corporation
F. T. Baker—IBM Federal Systems Division
Susan Voigt—NASA

PANEL OVERVIEW—Victor R. Basili

The development of correct, reliable, less expensive software continues to be a major problem. A great deal has been written and said about various techniques and methodologies for software development and how they are meant to aid in the development process. Unfortunately, most of the available material is by the author of the technique, be it individual or company. This does not always allow the outside user of the technique a fair appraisal or full understanding of how good the technique is, how to use it, and how to adjust it to his environment. This is true for several reasons. First, the author's experience is often limited to a specific application or set of applications and specific environments. There are some genuine questions that arise when taking a technique and moving it to a new application or environment that the developer of the technique had not anticipated. Second, the author does not always tell the prospective user everything he needs to know. Often this is just due to a lack of documentation, or a set of basic assumptions and background that the author did not realize was even necessary. Lastly, one cannot normally expect the author to emphasize the weak points or problems in the methodology. That is just human nature.

The purpose of this panel and the following set of papers is to discuss a set of techniques available in the open literature, some very new, some that have been around for awhile, and ask for an analysis and evaluation by current users. Each of the panelists is not the author of the methodology but a member of a company that is using the methdology or overseeing a contract on the use of the methodology. Some of the users are old hands at the technique; some are novices. I have asked each of the panelists to prepare a short paper covering a brief description of the technique and an evaluation of the technique in a real environment. Suggested ideas to be included in the paper and the oral presentation are given below.

I. THE TECHNIQUE
- The techniques you have been using.
- A description of the project you are using it on.
- The phase of the project in which it is used.
- The phases of the project it affects.
- An overview of the technique.
- Why you chose it.
- The extent to which you are using it.

II. EVALUATION OF THE TECHNIQUE IN A REAL ENVIRONMENT
- What have you felt are its good points and how they have shown to be good.
- What have you felt are the weak points and why.
- How you have adapted it to your environment.
- Would you use it again and if so how would you change or have you already changed the way it should be applied.
- What would you recommend to someone else applying it.

Certainly lots of techniques could have been covered but there was limited time and space available. The techniques chosen were based partially on my own interest and partially on the availability of people willing to discuss specific techniques. Discussants and topics include:

PSL/PSA—Donald J. Reifer,
SADT—Donn Combelic, ITT Telecommunications
Structured Design—Dr. J. A. Rader, Hughes Aircraft
Jackson's Methodology—Clifford M. Bernstein, EXXON Corporation
Boeing's IPAD Methodology—Susan Voigt, NASA Langley Research Center
Correct Program Design—F. Terry Baker, IBM Corporation

The methodologies deal with various phases of the software development process, from requirements to program development, some emphasizing one specific phase and some covering several phases. PSL/PSA and SADT deal predominantly with the requirements phase. Structured De-

sign, Jackson's Methodology, and Correct Program Design deal with various aspects of design and development. The Boeing IPAD Methodology covers the gamut from requirements to completed product.

What follows is a position paper for each of the panelists. Each is meant to be self contained, complete with references to the appropriate material.

EXPERIENCE WITH PSL/PSA—Donald J. Reifer

INTRODUCTION

This paper briefly describes the experience we have had in using the University of Michigan developed Problem Statement Language (PSL)/Problem Statement Analyzer (PSA).[1] We are using these computer assisted requirements tools to document and analyze operational flight software requirements developed for the Titan 34D segment of the Space Transportation System. The Titan 34D segment is providing real-time guidance, checkout and control requirements for implementation on the Interim Upper Stage. These boost phase requirements are highly time-critical and computation-bound. In addition, they must be documented in accordance to the Software Part I Specification format of MIL-STD-483.[2]

PSL/PSA DESCRIPTION

PSL is a machine processable language for expressing functional and performance requirements for a data system in a rigorous and uniform manner. PSL contains a set of simple declarative statements that allow you to name conceptual objects in a proposed system, describe properties of these objects and display relationships between them.[3] PSA is a software package that accepts PSL statements, analyzes each statically for correct syntax, generates a requirements data base from the input statements, performs consistency and completeness analyses on the data base and generates various kinds of reports and the requirements document. The command language with which users invoke the PSA services is also simple and user oriented.[4]

SELECTION CRITERIA

We selected PSL/PSA from several alternatives for several reasons. First, it is commercially available and supported. There is a commitment to keep the system current and fix errors. Second, it is well documented and good training is available. Third, it is operational on many large computing environments including the IBM 370, CDC 6000/7000 and Honeywell series. Lastly, it is currently being used by several diverse commercial and military users. The multi-user feedback has made the system mature faster than the alternatives. Readers are directed to Davis and Vick's paper[5] for an excellent comparison of PSL/PSA with other techniques.

We elected to use the tool on the Titan 34D application because we felt there was relative low risk inherent in their requirements. Titan 34D requirements are based on flight-proven equations and logic. We felt that the potential benefits justified the risk associated with using a new tool to document known and mature requirements.

EXPERIENCE

The ISDOS staff of the University of Michigan was retained by the Martin Marietta Corporation to install the PSL/PSA system on their CDC 6500 computer and conduct training classes. Installation began in April 1977. Several problems occurred as the package was adapted to the machine. First, the number of pages that resided in core had to be adjusted in order to make the system more efficient in terms of internal charging algorithms. Second, a machine-dependent executive routine that generated control instructions to do routine calling had to be developed. Training was held in the April/May timeframe with general orientation classes, user classes and maintenance classes being held.

The PSL/PSA system was then used to generate operational requirements for the Titan 34D segment of the Space Transportation System. The positive results of its use can be summarized as follows:

1. It forced the user to understand his problem and address it in a disciplined manner.
2. It documented requirements in a uniform manner and eased the task of document maintenance.
3. It assisted in the identification of errors primarily in the areas of incompleteness and inconsistencies.

The negative features of the system are as follows:

1. The system was too general to be used for a specific application. An internal groundrules document had to be developed to tell the user what to do and what not to do (e.g., limited number of attributes, naming conventions for decimal numbers, keyword conventions, etc.)
2. The PSL/PSA systems used large amounts of computer time that were not planned for.
3. The system is oriented towards a business environment and makes assumptions that make it difficult to describe requirements for realtime systems (e.g., concept of counters and interfaces hard to describe, notation is not compatible with scientific symbols, etc.).
4. The system was hard to sell to the users who were engineers. Therefore, changes had to be made to overcome user criticism (e.g., illegal characters like / in ft/sec had to be made legal).
5. The system had to be extended in order to provide I/O tables required by the B5 specification[7] format. A FORTRAN program was developed to search existing files and generate the tables.

The positive benefits resulting from use of PSL/PSA compensated for the negative experiences. Plans have been made to utilize the system to describe software requirements for other military systems.

ACKNOWLEDGMENT

The work reported in this paper was accomplished by the Martin Marietta Corporation under contract to the Air Force System Command's Space and Missile Systems Organization (SAMSO). I would like to thank them for their assistance in the preparation of this report.

REFERENCES

1. Teichroew, D. and E. A. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing," *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, pp. 41-48.
2. MIL-STD-483, Configuration Management Practices for Systems, Equipment, Muinitions and Computer Programs, 31 December 1970.
3. Hershey, E. A., et al., "Problem Statement Language-Language Reference Manual," ISDOS Working Paper No. 68, University of Michigan, 1975.
5. Davis, C. G. and C. R. Vick, "The Software Development System," *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, p. 74.
6. Johnson, L. *JSS Software Systems Engineering: Preliminary Evaluation of CARA*, Logicon Corporation, 1975.
7. MIL-STD-490, Specification Practices, 30 November 1968.

EXPERIENCE WITH SADT—Donn Combelic

BACKGROUND

SADT, Structured Analysis and Design Technique, is a registered trademark of SofTech Inc., Waltham, Mass. ITT has used the "Structured Analysis" part of SoftTech's SADT since early 1974. In mid-1975 we began to develop for real-time switching software our own structured design technique, called FP2 for Functions-Processes-Flowgrams-Programs, based upon precepts and syntax of Structured Analysis. Thus the technique we are now using for analysis and design is called SA/FP2.

GENERALITIES

The principal basic ideas of the technique are: determine the "what" before the "how", docomposition from the top down to reveal successive levels of detail, output in the form of diagrams each of which gives a limited amount of detail, each diagram is critiqued in writing by one or more persons other than the author of the diagram, needed information unknown to an author is obtained by interviews with outside experts. Each diagram is comprised of boxes that represent "activities" interconnected by arrows that represent "data" used by an activity for input, output and control. A box plus its arrows constitutes the "context" of that activity—it is that (bounded) context which is decomposed to understand and show more detail in a diagram at the next level.

OVERVIEW OF SA/FP2.

SA/FP2 is carried out in five phases, one of which is concurrent with two of the others. The first phase is that of Structured Analysis (SA); the remainder are those of design, that is, FP2. A brief summary of each of the five phases is given in the following paragraphs.

Structured analysis phase

SA is ideally applied to the total system. However, in most applications we have applied it only to real-time switching software. In such a case, the primary inputs are a list of customer requirements plus functional specifications of the telephone hardware of the system. The output is a Functional Requirements Model (FRM) in the form of a set of activity diagrams many of which are accompanied by a page or two of explanatory text and definitions of terms. The FRM shows what functions the software must contribute in addition to those of the telephone hardware in order to fulfill the customer functional requirements. To the greatest extent possible, software design considerations, such as data layouts, scheduling, priorities, handling of queues, buffers and computer peripherals, are kept out of the FRM. Thus the FRM emphasizes the "what," not the "how".

Transfer phase

This is the first phase of design. Its principal inputs are the FRM and the software design constraints. Typical constraints are: choice and arrangement of computers, computer peripherals, programming language, requirements for traffic handling, engineerability, extensions, maintainability, etc. The outputs of the transfer phase are a high level data layout model and a single level "action group" model. (A software action is defined as a sequence of instructions which, once started, can run to completion without waiting because all needed inputs were available at its start.) For each action group, a convenient set of one or more contiguous activities, along with the data arrows at the boundaries, is selected at an appropriate level of detail from the FRM and transferred (as a single box) to the action group model. The action groups are interconnected as the corresponding sets of activities were interconnected in the FRM. The high level data layout model is developed before and during the transfer procedure. The transfer phase is complete when all activities of the FRM have been accounted for in the action group model.

Action group decomposition phase

All action groups are decomposed to the level of individual actions. The output is a set of activity diagrams where each box at the lowest level of detail represents an individual action. An additional output is further detail of the data layout. During this and the preceding phase the decomposition rules are the same as for the SA phase, but the SA syntax is augmented to handle action starts and completions.

Flowgram phase

Each action is decomposed, according to its implicit control flow sequence, down to the level of individual routines, each of which appears as a separate box on the lowest level diagrams. The control flow is shown on each diagram in a special syntax, hence the name "flowgram." The output of this phase is a "flowgram model" for each action. The previous syntax is augmented to handle control flow. It turns out that when the control flow sequence and the individual routines are coded the resulting set is a structured program. Thus there is a structured program for each action.

Coordinator phase

The coordinator is that software which, among other things, starts all actions and to which all actions return upon completion. It thus includes the functions of scheduling, handling of queues and management of memory. It is convenient to include within the coordinator the treatment of interrupts and the handling of telephone and computer peripherals. It is interesting to note that none of these functions relate directly to the customer's basic functional requirements, rather they all depend on the nature of the system. The functional requirements for the coordinator begin to emerge as early as the SA phase, become more clear by the end of the transfer phase, but cannot be known completely until the action group decomposition phase is finished. By that time the coordinator can be completely specified and designed. Note that the techniques described for the preceding phases can be applied to the analysis and design of the coordinator.

EARLY EXPERIENCE—1974-1976

Development of the FP2 design methodology reached the point where it could be used in practice only at the beginning of 1977. Thus all our prior experience was limited to Structured Analysis as taught by SofTech and refined by ITT and SofTech together. We adopted SA in early 1974 for two main reasons. First, it provided a disciplined way of understanding requirements in detail before starting design. Second, it offered a method which promoted teamwork. The latter was a particularly difficult problem on some of our projects in Europe where a team would consists of members with widely varying experience from up to eight different ITT companies speaking six different languages. Of the approximately twenty ITT projects where SA has been used, all but one are in Europe.

Strong and weak points

A partial list, derived from our early experience with SA, is as follows:

- SA estimated to decrease overall software development cost by at least 20 percent and significantly improve software quality—estimated 2 to 10 times less bugs found during integration testing, varies with project.
- Very definitely promoted teamwork.
- Hard to think all the time in purely functional terms.
- The written comments (by other than the author) required for each diagram resulted in continual review, in effect "walkthroughs." (Note : Commentators should normally be other authors in the same team.)
- Interviews of outside experts proved efficient method of obtaining specialized information.
- Forced making high level decisions early, thus providing a sound basis for later lower level decisions.
- Encouraged agreement on requirements before start of design.
- Lack of follow-through on design methodology (later overcome by FP2) was bothersome.
- Permitted non-software people to understand the contributions of software functions to system operation.
- Much more useful information per sheet (diagram) than with documents in prose.
- Provided easy way of measuring progress during analysis.
- SA excellent for many applications other than real-time switching software, but space does not permit elaboration.

Some mistakes made and lessons learned

A partial list follows:

- Method was oversold in the beginning as a panacea.
- Proper use of SA requires a fundamental change in mental outlook—difficulties of achieving such a change were underestimated.
- Mere "training" is inadequate—"education" is required.
- Potential authors must be selected on the basis of intelligence and willingness to try a new way rather than purely on experience.
- Constructive critics of the method are to be cherished, but destructive critics must be eliminated from the SA group as soon as they surface.
- Method takes much more time before design starts than previous methods. Inevitable if requirements to be thor-

oughly understood and agreed before design, but causes impatience in some participants and management.
- Need one "friendly" group to try method first. Afterwards it's better to "offer" the method to other groups than to try to "sell" it—hard sell doesn't work.
- "Discipline" in use of method is very important: syntax, conventions, rules, author-commentator cycle, completion of a level of decomposition before going to next, each diagram must increase understanding, etc.
- Large amounts of paper generated. Project librarian must be assigned, as recommended by SofTech.
- Formal training is lengthy—two to three weeks full time—but necessary.
- Potential authors must have a project in mind during training and be assigned full-time to it immediately thereafter.
- Follow on assistance by a trained "monitor" is obligatory during initial application of method.
- Monitor must confine his attention to proper use of the method rather than become involved in the substance of the analysis.
- Training plus monitoring by SofTech is costly: 20 to 40 thousand dollars per course for up to ten authors, but worth it.
- Ideal team during SA phase for our type of large switching projects seems to be two or four persons each from systems, hardware and software.

Recent experience—1977

It is our estimate that about two-thirds of the value of SA/FP2 is in the SA part. Nevertheless, the availability of a follow on design method, in our case FP2, makes the application of SA easier because the SA authors are more ready to defer design considerations to the FP2 phases.

The development and acceptance of FP2 has proved extremely difficult and its success has not yet been fully demonstrated. We wanted to use the same basic precepts and syntax as in SA so that the designers, some of whom will have also participated in the analysis by SA, would not have to learn a new syntax and set of principles—we wanted a sort of "continuous" method, starting with a list of requirements and constraints and ending up with detailed coding specifications. This "continuity" and similarity of syntax is important to software maintenance personnel and will assist comprehension by interested customers.

This section concentrates on our 1977 experience with FP2; subsequent experience will be covered during the presentation at NCC 78.

- Underestimated importance of providing detailed guidelines for carrying out the Transfer phase.
- Initially called the output of the Transfer phase, the "process model." "Process" has many meanings and caused great confusion. The neutral phrase "action group" conveys the intent without confusion.
- Software design still requires great skill, but FP2 permits easy comprehension after key design decisions have been made.
- FP2 criticized for not rendering high level software design "semi-automatic" or at least "semi-algorithmic."
- Direct coding from flowgrams is in most cases straightforward and can be done by programmers other than the designers.
- The fallout of a "structured program" for each action has proved very appealing.
- Test plans can be developed throughout FP2 in increasing detail and related directly to diagrams.
- Much debugging is done by reference to flowgrams instead of listings.

REFERENCES

1. *An Introduction to SADT*, SofTech, Inc., Waltham, MA, document 9022-78, Feb. 1976.
2. Ross, D. T. and K. E. Schoman, Jr. "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 6-15.

EXPERIENCE WITH AN APPLICATION OF STRUCTURED DESIGN—J. A. Rader

INTRODUCTION

The application of structured design to a 20 man year project which generated 100,000 lines of code is described. Included are a description of the project, productivity figures, and a discussion of strengths and weaknesses of the technique as practiced on the project.

THE APPLICATION

Introduction

The Computer Aided Design (CAD) Department in the Hughes Aerospace Groups contains about sixty employees. The department provides Computer-Aided Design/Test/Manufacturing software and services; it operates and supports a DEC system 10 computing facility; and it operates and maintains a Gerber photoplotter and several digitizers.

Most of the software is data manipulative in nature—files are read; fields are extracted from records and massaged; arrays are built, operated on and sorted; reports are generated; and data bases are accessed and updated. The primary language has been and continues to be FORTRAN. In addition, there is an extensive library of FORTRAN-callable assembly language routines to perform bit and character manipulations as well as other special functions. Where very heavy CPU utilization is expected, assembly language is also sometimes employed.

Conversion decision

Several years ago, a corporate decision was made to phase out the Honeywell G635, the computer on which the CAD System at that time ran. Among the many alternatives considered for rehosting the CAD System, the one finally chosen was to purchase a DEC system-10 and convert the CAD system to run on that computer. The primary reason for selecting the DEC-10 was a proven time-sharing capability.

The conversion from the G635 to the DEC-10 was a conversion only from the standpoint of function.

Existing programs were inventoried and for each it was decided which would be converted essentially as is, which would be modified, and which would be discarded.

Goals and advanced plan

As we started to plan, we recognized that it was very important to firmly establish our goals, and to determine very specific milestones. Thus we would be able to measure our progress and to report on it to our management, who was picking up the tab.

Succinctly stated, the major goals were: (1) to create a unified system tied together by a central data base; (2) to create software that was reliable and maintainable; (3) to provide a user interface that was easy to use and that was consistent across all software; and (4) to proceed in a manner that allowed us to measure how well we were meeting schedules.

In late 1973, an advanced planning and development activity was formed. The advanced planning group was to define the overall structure of the system, and to specify the procedures to be followed in specifying and implementing the system.

In November of 1973, the author attended a 6-day in-plant seminar on structured design. This seminar was taught by Larry Constantine and proved to be of immeasurable value. The value arose not from revolutionary concepts but rather from a well reasoned and coherent discussion of the relevant concepts. The ultimate result of attendance was the generation and documentation of a methodology for practicing structured design in our organization. This methodology is described in the next section.

Outputs of advance group

In the approximately 18 months of its existence, the advance group produced 4 basic outputs. First, it produced a system concept. Second, it defined the standards and procedures to be followed in rewriting the system. These were documented in a Standards and Procedures Manual issued to all programmers. Third, it defined most of the applications support software. Fourth, it provided a test vehicle for the standards and procedures defined, and provided the first productivity figures for the methodology. These figures were used in estimating the effort required to implement the main body of CAD software.

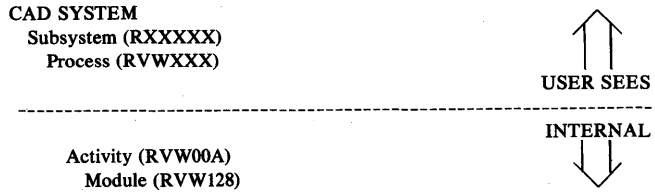


Figure 1—Software hierarchy

STRUCTURED DESIGN

The following hierarchy, illustrated in Figure 1, is used to identify levels of CAD software: system, subsystem, process, activity, and module. System means the whole CAD System. A subsystem is a major functional area. Examples of subsystems are routing, digital test and simulation, design capture, etc. A process performs an identifiable user function which may be simple or complex. The process level is the lowest level visible to the user.

A process consists of one or more activities, where an activity is an executable core image or job step. A module, which is the lowest level, is a single FORTRAN or assembly language subroutine. Our FORTRAN modules contain an average of 35-45 statements. Assembly language modules have an average length of 50-60 statements.

Each activity and module is uniquely identified by a six character code. Examples are shown in parentheses in Figure 1. Thus module RVW128 would be module 128 in the View (VW) Process of the Routing (R) Subsystem.

Structured design, a la Constantine, is applied either at the process or activity level, depending on the complexity of the process and constituent activities. Design documentation always includes a structure chart (hand drawn, as generated in the design process) and a completed module description form (MDF) for each and every module. A module description completely defines the *function* of a module so that it can be coded from this documentation alone, with perhaps reference to appropriate data structure documentation.

For an easily grasped function, e.g., the binary search of a single precision array, a description of the calling sequence may be adequate documentation. However, it has been our experience that many modules require either flow charts or pseudocode for unambiguous documentation. This is true even though the emphasis is on the function not the detailed coding of the module. Typically, this situation obtains for the higher level modules in a structure chart. At that level it is frequently the case that the function of the module and the flow of control within the module are not really separable.

RESULTS OF APPLYING STRUCTURED DESIGN

Productivity figures

The advanced planning activity implemented 336 modules of applications support software. These modules contained

	Advance Group (336 Modules)	Main Development Effort (2200 Modules)
Man Hours Per Module		
Specification	3.3	5
Structured Design	3.0	5.4
Coding and Integration	8.8	9.8
Total	15	20

Figure 2—Manpower statistics

11,104 lines of code, an average of 33 lines/module. There were 180 FORTRAN modules (average length 25 lines), and 156 assembly language modules (average length 42 lines).

The manpower figures were specification and analysis, 28 MW (man weeks); structured/implementation design, 25 MW; and code and integration, 74 MW. Figure 2 displays the corresponding figures per module.

The largest activity in this collection of software contained 117 modules (3551 lines), of which 105 (2936 lines) were FORTRAN. The distribution of calendar time for this program was: specification and analysis, 26 weeks; structured design, 6 weeks; and coding and integration, 20 weeks. Thus, although specification and analysis only accounted for 22 percent of the manpower it consumed 50 percent of the calendar time.

Column 3 of Figure 2 summarizes the figures for the bulk of the CAD software. This data represents approximately 18 man years of effort including supervision time. Accounted for are 27 processes, 53 activities, and over 2200 modules.

Problems encountered

Although our experience with structured programming has been strongly positive, we did encounter some problems. Moreover, looking back, we see areas where improvement is needed.

The biggest need for improvement has to be in the area of specification. Once a good specification has been generated things become very manageable. However, we have found it extremely difficult to write a specification which on the one hand a user can read and understand, and which on the other hand defines things well enough to allow design to begin in earnest.

A second problem area is related to one of the design goals. From the start it was impressed upon the programmers that they were to put design before efficiency. As a result a couple of activities were implemented which were very much more expensive to execute than they should have been. These subsequently, had to be modified for efficiency.

In most cases this efficiency problem might well have been avoided had we more strictly followed one of our own published procedures. We did not require each module to be reviewed by another programmer as we said we would. The excuse tends to be "we just don't have the time", and is used by programmers and supervision alike. In the face of tight schedules, this excuse is not easily dismissed. This problem will doubtless be struggled with for some time to come.

A final problem is the difficulty in training enough programmers to be good designers. The training problem is particularly troublesome because there is no way to give years of experience, and the attendant design maturity, to even a highly capable junior individual. This is important because the structured design of a large process requires the judgment to make numerous decisions, which involve tradeoffs between strict adherence to structured design principles, on the one hand, and effective use of human and machine resources on the other. The best solution is to employ the most qualified designers on critical designs, and to use less critical designs for training. But when schedule constraints force many important designs to overlap, less desirable compromises have to be made.

Summary of benefits

Although we did not have any solid prior productivity data, we definitely feel that structured design has improved productivity. This, however, was not a goal in our adopting structured design, but has been just a happy side-effect. What was anticipated were increased reliability, increased maintainability, and increased visibility.

There is no doubt that we have achieved increased visibility. Moreover subsequent experience with modifying structured programs convinces us that increased reliability and maintainability have been achieved.

It was found that structured design allowed us to make very good use of personnel. We have been able to really load up an activity in the module coding and checkout phases without introducing confusion. It has also been easy to quickly move a programmer from one activity to another, with little loss in effectivity. Moreover, we have gotten excellent productivity from beginning programmers.

Only mild reluctance to adopt structured design techniques was manifested by the staff. Junior personnel adopted easily with no apparent "loss of individuality" response. There was some initial thrashing with senior personnel as we all strove to understand the implications and tradeoffs of modularity. For instance, not everyone accepted at first that structured design was indeed distinct from what they were already doing.

A final word on interpreting our experiences with structured design. We feel that our experience is unique to our application and our environment. A different application in a different environment might yield better or poorer results. Nonetheless, we are confident that, for most applications, structured design will yield more reliable and maintainable systems, while providing good visibility of the design and implementation processes.

BIBLIOGRAPHY

1. Constantine, L. L., and E. Yourdon, *Structured Design*, Yourdon, Inc., New York, (February 1976).

2. Rader, J. A., *Structured Design—A Case History,* Infotech State of the Art Report on the Practice of Structured Design, London, (1976).
3. Stevens, W. P., G. J. Myers, and L. L. Constantine, Structured Design, *IBM Systems Journal,* No. 2, 1974, pp. 115-139.

EXPERIENCE WITH EXXON'S IMPLEMENTATION OF THE JACKSON PROGRAM DESIGN METHOD —C. M. Bernstein

In 1973, Exxon's Mathematics, Computers and Systems Department conducted an evaluation of the new technologies of program development. The project was motivated by the increasing manpower cost for software development and maintenance and the increasing business vulnerability to software failure. We concluded that a program's structure is the key to its effective development, enhancement, execution and support. Without a program's structure we could not reliably construct the program in parts, put the parts together such that their interactions would be predictable, and have the whole structure achieve its specified purpose.

Michael Jackson, now of Michael Jackson Systems Ltd., was then an ACM lecturer. We found that he had a teachable method for the logical design of structured programs and a precise notation to express them. In addition, his method was effective at attacking practical programming problems where the designer is not free to define input and output formats and user interfaces. Michael taught three program design courses at our Florham Park, New Jersey offices in the winter of 1973. The courses were well received and, together with Michael's consulting on specific applications, provided the knowledge we required. We adapted Jackson's Methdology to our environment and renamed it Program Structure Technology, PST.

It is important to remember that PST does not address the systems design process. PST is not concerned with defining the organization or contents of files, specification of input and output formats or transactions, designing data bases, or defining required processing. PST is concerned with the work of designing and implementing the program which meets those specifications. PST incorporates the technologies of top down design, structured programming, top down development and test, and structured walk-thrus.

We conducted our first in-house PST course in August of 1974 and aggressively fostered the assimilation of PST over the following three years. Over 1,000 programmer/analysts have been taught PST. All Exxon regions can now provide their own training and support. This includes training for supervisors, consulting, and the support of standards. PST is being used in both large and small installations to develop batch commercial and interactive data base applications, as well as minicomputer, process control and program product software.

The Jackson Program Design Method defines hierarchical program structures whose components are dissected. There are four basic component types:

Elementary, which are not dissected
Sequence, whose parts are executed once each, in order
Selection, one of whose parts is executed, the choice part depending on a condition
Iteration, which has only one part which is executed zero or more times

The four component types have exact analogies in data usage structures for files, records and in internal data. The program's structure is designed to correspond to the usage of the data to be processed. Alternative usages of data can be imposed on a given set of data and the alternative program structures evaluated. Each operation carried out by the program appears in an appropriate component. For example, "INITIALIZE CUSTOMER TOTAL" should appear in a component which happens once per customer. If no appropriate component exists, the program structure is deficient. Where there is a conflict between data usages to be processed by one program, the program is designed as if it were two or more separate programs. The separate programs are subsequently combined by the technique of program inversion. Where a data usage cannot be handled simply, more elaborate forms of iteration and selection must be used.

The major benefits of the Jackson Methodology are as follows:

Reduced program complexity
Eliminated logic errors at design instead of debugging in the testing stage or later
Identified sensitive points in the problem specification
Easily maintained programs
Effective program documentation as a by-product of design
Step-by-step methodology has enabled effective use of software to support the process

Four PST projects of varying size (.5, 1, 5.1, and 25 work years of effort) were evaluated and productivity was found to be above 7000 lines per work year in every case. This compares favorably with the New York Times Archives project's 9000/WY and the "Industry Averages" of 2000-4000 lines/WY. In the case where we attempted to measure program quality, we found less than one error per program during the first six months of production.

We have done little to the Jackson Methodology to adapt it to the Exxon environment other than minor changes in terminology and emphasis. We have complemented it with our own material on structured programming in PL/I, top down development and test, and structured walk-thrus. We have also employed different pedagogical techniques to enable programmers who are not experienced instructors to teach the PST course.

The Jackson Methodology has been extremely successful in Exxon and is now virtually a standard throughout the world. I recommend not underestimating the difficulty of teaching programmers "how to program properly" and develop an assimilation plan and a skilled staff to accomplish it.

REFERENCES

1. Jackson, M. A., *Principles of Program Design* Academic Press, 1975.
2. Pinsonneault, L. L., Course Overview. *Program Structure Technology Course Manual,* March 1975.

INITIAL EXPERIENCE WITH A METHODOLOGY FOR CORRECT PROGRAM DESIGN—F. T. Baker

In the forthcoming book, *Structured Programming: Theory and Practice,*[1] the authors describe three techniques which have been incorporated into a Methodology for achieving correct designs. These are:

1. A view of program correctness as a demonstration of a correspondence between the function of a program design (i.e., the set of ordered pairs corresponding to input states and output states) and the function required by its specification. This approach, when used with stepwise refinement, permits selective and incremental correctness proofs to be carried out, since it incorporates a procedure for verifying the correctness of the expansion of a specification into any one of a basic set of control structures. (The expansion of a specification at any level into a program design can thus be verified, contingent on the correctness of lower-level specifications and their expansions.) Furthermore, proofs can be carried out with varying levels of rigor, ranging from a set of questions the designer may use to validate an expansion to a formal demonstration recorded in a precise manner.
2. A method for incorporating specifications into program designs to support correctness demonstrations when desirable. Each specification (either initial, or those generated in the expansion process) is retained as a comment (logical commentary) directly associated with the control structure which refines it.
3. A design language (Process Design Language) to assist in the design process and to record the history of a design. PDL includes a standard "outer syntax" of essential control and data structures, and encourages development of an "inner syntax" appropriate to each design environment.

Figure 1 is an example of a design for a program which is to save the maximum value occurring in an input sequence. In that design, each of the paired brackets encloses logical commentary which is a functional specification for the control structure which follows it (sequence, ifthenelse or while do in this case). For each of the structures the appropriate proof procedure can be carried out to demonstrate the correspondence between its specification function and its program function. Furthermore, this can be done prior to the completion of the expansion process, or even on selected portions of the design.

The methodology is primarily aimed at the detailed design of a program. It covers the period between the formulation

```
[m:=maximum(inputseq)]
do
  [(inputseq=empty→m:=undefined |
        inputseq≠empty→m:=next(inputseq)]
  if
    inputseq=empty
  then
    m:=undefined
  else
    m:=next(inputseq)
  fi
  [m:=maximum(m,remainder of inputseq)]
  while
    inputseq≠empty
  do [m:=maximum(m,next(input)]
    temp:=next(input)
    [m:=maximum (m,temp)]
      if
        temp>m
      then
        m:=temp
      fi
    od
  od
```

Figure 1

of an effective system design, and the translation from the design language into an implementation language. It was developed to introduce more precision into the design process and to encourage more consistent expression of designs. Whether or not formal correctness demonstrations are carried out, the stress on viewing programs as functions, developed from specifications through a rigorous refinement process, should help achieve the goal.

Experience to date suggests that the methodology is capable of being practiced in the application development environment. We believe that the control structures inherent in PDL are sufficient to support all levels of the design process, from system and module specification down to precise algorithms. The invention and use of logical commentary direct attention to specification and program functions, as they were intended to do. In particular, they encourage the designer to specify and deal with boundary conditions and anomalies which frequently are poorly attended to and which sometimes lead to difficult-to-find errors. Finally, the view that each progrm should be designed as if it is to be proved correct, means that even if correctness demonstrations are not formally carried out, the program has a greater likelihood of properly embodying its specification.

Experience to date also indicates several areas where more work is needed. The nature of the expansion process, and the desire to record the design history, mean that much copying is done as the design is developed. An interactive support tool appears useful to assist designers in this expansion and recording. The data structures in PDL (stacks, queues, sequences and sets) are useful ones, but better proof techniques to validate operations with them must be developed. There is a notational problem inherent in specifying functions precisely, particularly in nonmathematical envi-

ronments, which must be solved through a combination of inventing better notations and abstracting operations. Finally, the ability to demonstrate correctness does not mean that it is appropriate to do so in all cases; better guidelines for applying the varying degrees of rigor possible in the methodology must be developed.

REFERENCE

1. Linger, R. C., H. D. Mills, B. I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, 1978, to appear.

EXPERIENCE WITH THE IPAD SOFTWARE DEVELOPMENT METHODOLOGY—Susan Voigt

NASA is sponsoring the development of a computer-aided design system for use by the aerospace industry. The system, denoted IPAD, is being designed and implemented by Boeing Commercial Airplane Company and Boeing Computer Services. IPAD is a software system to enhance the computer complexes of aerospace companies to improve speed, efficiency and reliability of the design process for complex aerospace vehicles. The contract calls for application of an effective software engineering approach to minimize programming and software design errors, as well as to produce highly portable software.

THE TECHNIQUE

NASA established general guidelines for phases of the development and release of software in stages, to allow early user testing and experience. The development phases are:

1. Definition of the Problem (namely the Aerospace design process)
2. Requirements Definition (integrated information processing and functional requirements)
3. Development of IPAD System Specifications
4. Preliminary Design
5. Detailed Design, Code, and Test for each incremental release
6. Acceptance Test and Demonstration with sample problems for each release
7. User Training and feedback
8. Software maintenance during remainder of development contract

Several plans and specific documents were called for in order to encourage a systematic approach and a well documented product:

1. Management and Technical Plans to describe general approach
2. Configuration Control Plan to control and track all changes to requirements, design, code and documents
3. User Involement Plan to insure the sytem developed is satisfactory to the users
4. Test Plans to establish systematic procedures for development and acceptance testing
5. Software Standards Handbook
6. Requirements and Preliminary Design Documents
7. Preliminary User's Manual written during design so early user feedback can be obtained
8. User and Demonstration Manuals
9. Installation and Maintenance Manuals

As the basis for the IPAD software engineering methodology, the Boeing IPAD Development Team selected the Boeing Computer Services "Systematic Software Development and Maintenance (SSDM)" approach. SSDM is basically a set of general guidelines for all phases of the software life cycle, and it corresponds well with the NASA requirements.

An Industry Technical Advisory Board (ITAB) was established at the start of the contract to closely involve the prospective user community. They have helped review and critique the requirements definition and software design phases. Subsequently, they will have the opportunity to install and test the software at their own computing facilities.

EXPERIENCE AND EVALUATION

At the time of this writing, the development is in phase 4, Preliminary Design. The techniques used to date and an assessment of their usefulness in the software development process is described below.

In phase 1, a reference aircraft design process was documented in flow diagrams indicating activities and decision points, with accompanying discussions. Also communications between various disciplinary groups participating in an aircraft design project were diagrammed to illustrate the complex network of interfaces. Separate volumes were written to document the interactions between designers and the manufacturing organization and the activities in managing a product development. These three volumes written by engineers representing potential users defined the problem to be addressed with the IPAD system.

The requirements were defined by the engineers in phase 2. The BCS technique SAMM (Systematic Activity Modeling Method) (Reference 1) was used to chart the inputs and outputs (description and quantity) for each activity in the flow-charts of phase 1. The user's view of his requirements for computer-aided support in the aerospace design process also was documented.

The results of phases 1 and 2 represent a very thorough definition of the aerospace design process and CAD users' needs. These were well-received by the user community and have provided guidelines for their own analyses within their respective companies. The flow diagrams of the design process and the SAMM charts of the data flow are well correlated and provide a very systematic look at the problem.

Phase 3 was done by the software team, assisted by the engineers. They developed a concise set of IPAD requirements based upon the engineering documents and NASA requirements for the IPAD system. A formal analysis checked that each requirement was complete, correct, unambiguous, precise, consistent, relevant, testable, traceable, free of unwarranted detail, and manageable. The engineering team developed criteria for acceptance testing for each requirement. Each test criteria was summarized in a paragraph which accompanies the requirement statement in the IPAD Requirements Document. The requirements were reviewed carefully by both NASA and ITAB, with considerable feedback and revision resulting. A set of IPAD system specifications were not produced, per se; the IPAD Requirements became the baseline for further development.

The formal analysis of requirements was not successful. Ambiguities, inconsistencies, and redundancies were very difficult to eliminate, especially between similar requirements. These arose from using secondary sources in developing the IPAD requirements and giving them weight equal to the primary source, the engineering definition of the problem. A satisfactory set of requirements was obtained through careful reorganization and joint review by software and engineering team members and NASA. The inclusion of the summary for acceptance test was very helpful in clarifying the intent of a requirement, as well as setting the stage for later testing. It also forced the engineering and software teams to collaborate, and is highly recommended for future projects.

In Preliminary Design, phase 4, a user interface model was developed using state transition diagrams (Reference 2). This included a set of user functions correlated to the IPAD requirements. These state diagrams have been used to walk through "user scenarios" to illustrate the functioning of IPAD from a user point of view in performing a specified set of tasks. The other major components of IPAD are: the executive, the information processor, and the other systems interface and are currently undergoing design by separate subteams. Coordination among the various subteams in producing an integrated design has been difficult.

While the approach used has been successful in achieving a good set of requirements, IPAD is not yet developed and the design methodology is unproven. The basic concepts appear to be an effective approach and further assessments can be made when the software is developed.

REFERENCES

1. IPAD Document D6-IPAD-70012-D, "Integrated Information Processing Requirements," June 24, 1977 Boeing Commercial Airplane Co., under contract NAS1-14700. (Data Modeling Method, SAMM)
2. Parnas, David L., "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System," *Proceedings, 24th Conference ACM*, 1969, pp. 379-385.

Area Director:
Jack Goldberg
SRI International
Menlo Park, California

Formal methods in programming and microprogramming

Two of the three sessions in this area are about microprogramming, and the third is about programming in general. The dominant theme in all the sessions is the current attempt by researchers and practitioners to make programming a formal process. Roughly speaking, this means the use of mathematical rigor in defining, specifying, translating, transforming, analyzing and verifying programs (including microprograms).

The work is motivated not by a love of formal methods for their own sake, but by hard economic objectives. Users are now demanding increased confidence in the adequacy and reliability of delivered software, greatly reduced cost of production and maintenance, and greatly increased flexibility with respect to change, growth and transportability. These demands are not being met by the informal techniques that have accumulated over the years. The work on formal methods is intended to make major advances in the way software practice meets these demands.

With few exceptions, the old techniques—and the tools that support them, have aimed to help the programmer to describe, encode, recode, package and probe his large and complex product. The task of fully understanding the product and of ensuring that it and all the steps used in its production are correct, has been left to the human judgement of producers and consumers. With some serious exceptions, present practice has been generally successful due to the dedication of computer professionals, the effectiveness of social processes in dealing with errors in output data, and, of course, the great economic benefit of using computers for solving problems. This success has come at high costs that are no longer acceptable. Furthermore, new applications and market conditions have arisen that bring very high penalties for error, inflexibility and inefficiency.

The response of the computer science community has been to attempt to put mathematical rigor into the task of understanding what a program does (its "semantics"), or, in linguistic terms, what is the meaning of its text.

Of course, understanding precisely what a program does implies knowing precisely also what the translators, optimizers and interpreters do that may operate on the program. If the description and analysis of programs and their processors could be put on a firm mathematical basis, and if there were effective procedures for carrying out formal analyses, the consequences would be very significant. For example, one could build powerful optimizers, and be confident

that they would not change the meaning of a program. Also, one could verify that a program implemented a requirement for all possible values of data in the intended input domain, without having to use test data samples (a most costly and unsatisfactory process if high confidence is demanded).

As long as ten years ago, theoretical results showed that a formal basis for programming could be constructed. Since then much effort has gone into creating methods of formal specification and analysis, and tools to implement the methods, that are effective enough to be incorporated in software engineering practice. At the present time, several methodologies are reported to be in practical use in special application domains such as microprogramming, or on an experimental basis, for general programming. The methodologies require skilled practitioners. Because of this, one cannot say that a methodology is "ready" or "not ready" without saying who is going to use it. Nevertheless, it is claimed that there will be enough practitioners of the requisite skills to make a very significant economic impact on software production, especially as the methodologies mature.

The papers and panel sessions of this area give the attendees an opportunity to judge how valid these claims are, and to assess the relevance of the ideas and tools of the new methodologies to their own practice.

An approach to firmware engineering*

by DAVID A. PATTERSON
University of California
Berkeley, California

INTRODUCTION

Although microprogramming has only been a research topic for the past 10 years, the concept is nearly as old as computers.[1] Wilkes originally defined microprogramming as a systematic and orderly approach to the design of the control section of a computer. Rather than using an *ad hoc* approach with counters and decoders to generate control signals, Wilkes proposed that the control be organized as a matrix. This memory like approach to control remained largely of academic interest until the late 1950s. The first large-scale microprogrammed computer was the IBM 7950 in 1961. However, it was the introduction of the IBM System/360 line (with all but the largest system microprogrammed) that revived interest in microprogramming. The advance in speed and the reduction in cost of read-only memory (ROM) technology transformed microprogramming from an academic concept of mild theoretical interest to a practical approach to the design of commercial computers.[2] IBM was able to offer several internally different models with the same instruction set in order to provide a spectrum of cost and speed through the use of microprogramming. With the success of the 360, several microprogrammed computers were and are being built. In the late 60's some manufacturers made microprogramming available to the users. Nearly all computer manufacturers have at least one computer implemented via microprogramming, and the expanding list of manufacturers that provide user microprogrammable computers include Burroughs (E1700), Control Data Corporaton (Cyber-17), Data General (Eclipse), Digital Equipment Corporation (PDP-11/60) and Hewlett Packard (HP-21MX). This has expanded the potential domain of microprogrammers from computer architects to systems and applications programmers.

While microprogramming and "ordinary" programming have existed for almost the same amount of time, little of the considerable research in programming languages and methodologies has impacted the microprogramming field. Unfortunately, the following statement is a fairly accurate description of the state of the microprogramming:[3]

> "At present, microprogramming is an elite activity, performed effectively only by a small number of expert practitioners. The work is detailed, precise, time consuming, and considerably more expensive than present-day software programming. But, computer manufacturers have found they can get dramatic improvements in system performance by converting software into microprogrammed form."

An approach to expanding the group of practitioners of microprogramming is to reduce the complexity and expense of firmware by providing tools similar to those that have reduced the cost of software. The Strum system was designed as a prototype to explore the benefits and detriments of such an approach.

SOFTWARE ENGINEERING TECHNIQUES

The most obvious technique to enhance the production of microprograms is to provide a high level microprogramming language (HLML). One reason HLMLs have not been used is the importance of efficient firmware. With restricted size and execution times being primary characteristics of microprogramming, some doubt whether a HLML could ever be useful.[15] With the newer computers control memory has expanded (e.g., 16K control memory of the HP-21mx) thereby relieving some of the concern about size. A more important aspect is the influx of applications microprogrammers who have a very different set of constraints from the microprogrammer of the native instruction set. In the latter case one may be able to justify doubling the effort to reduce execution time by 10 percent. In the former case it is likely that ease of use, development cost and maintainability are much more important than small gains in execution speed. The necessity of a high level microprogramming language is further illustrated by the Figure 1. It is widely accepted that the programmer spends half his time testing the program. This figure indicates that a microprogrammer may spend half his time coding the microprogram. As it is widely accepted that a high level language will reduce coding time it makes eminent sense to provide a high level microprogramming language if it can produce reasonably efficient code.

Another technique that has reduced the time to develop software is structured programming. This phrase is so widely used that the criteria that determines whether a program is

* This research was supported in part by the Department of Energy under Contract EY-76-S-03-0034, PA214.

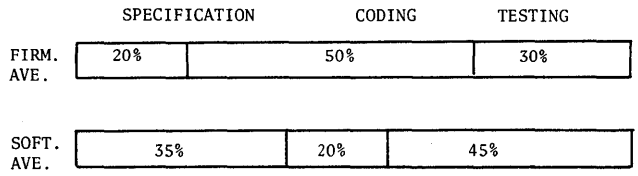


Figure 1—Percentage of time spent in specification, coding and test phases of microprogram and program development (from Reference 4).

structured is vague. By structured programming we mean a disciplined approach to programming that starts with the highest level of abstraction and proceeds with a topdown hierarchical development of levels of modules that are successively refined and expanded until the bottom refinement is the program in some programming language. A key aspect of this approach is for every module at each level to be "intellectually manageable," i.e., restrict the size and flow of control to create modules that are easy to read and to understand. Although assembly level programming does not preclude structured programming, it is easier to follow these concepts if the programming language supports this methodology. We believe that a cleverly designed microprogram language can hide the "unstructured" aspects of microprogramming plus provide tools conducive to structured microprogramming.

A final aspect of software engineering is validation. Here an attempt is made to determine whether a program is "correct." Unfortunately testing does not guarantee that a program is correct; it only means that someone (probably the creator of the program) is satisfied that the program preforms properly for some subset of the total input domain. As Dijkstra states:[5]

> "Program testing can be used to show the presence of bugs, but never their absence!"

If we conclude that testing is inadequate, our only alternative is formal program verification. This work began with the inductive assertion method of Floyd[6] that was elaborated by Hoare and Manna. This method first requires the addition of redundant text called assertions. Assertions are nonexecutable comments that describe the desired state of the variables of the program. These assertions can then be mechanically combined with the program text to produce logical formulas. If proven true, these formulas show a kind of consistency, usually called partial correctness, between the text and the assertions.

It is not clear that program verification is a practical tool for validating microprograms. In fact, there is a great deal of skepticism of the practicality of proofs of correctness in any environment.[7] The criticism is that only "toy" programs are proved, i.e., programs that use only integers or vectors of integers, deal with numeric algorithms (e.g., division), and are relatively short. Using these measures, one would have to conclude that microprograms are "toy" programs. Microprograms deal with integers (usually two's complement binary numbers) or integer vectors (memories), deal with numeric algorithms (instruction sets), and are relatively short (frequently less than 1024 words). Thus, the application of correctness proof techniques to microprograms may prove beneficial.

STRUM SYSTEM

The prototype system developed to test this approach to firmware engineering is called Strum (from STRUctured Microprogramming). This system includes a Pascal-like high level language, an optimizing compiler that produces microcode for the Burrough's D-machine, and a verification system.[4,8] The Burroughs D-machine is a general purpose microprogrammable, multiprocessor computer.[9] The computer was designed by the Burroughs Advanced Development Organization in 1968-1969 to be used in future Burroughs systems. The D-machine has several innovations. It was designed to be used in multiprocessor configurations. The arithmetic section of the machine is expandable in 8-bit increments from 8 bits through 64 bits. It was the first machine to use two levels of control memory (micromemory and nanomemory) to try to obtain the advantages of horizontal and vertical microprogramming. And at the same time it was built, it was one of the few computers to offer a control memory that could be altered by the user during program execution. The D-machine allows up to 4096 56-bit microinstructions.

THE EXPERIMENT

As a meaningful test of the system we decided it was essential to emulate a real computer. If a paper computer were to be emulated one might wonder whether the design of the new instruction set had not been compromised in order to avoid difficulties with the high level language or to simplify the verification of the emulation. Although this test would be emulation of a real computer on a real microprogrammable machine, some would wonder whether program verification, high level languages, and structured programming would not lead to correct yet inefficient code. As speed and size are very important in the microprogramming environment, it is possible that the overhead incurred using such techniques makes this approach unusable in the "real world."

To overcome these possible objections, an emulation of the same real computer was done twice. The first version was implemented via techniques traditionally used with microcode on the D-machine, i.e., assembly level coding, debugging the microcode on the machine, and testing the microcode using programs written for the machine being emulated. The other version was written using techniques proposed in this paper. Unfortunately it is difficult to create tests that are free from the appearance of bias. If one person did both emulations, one could correctly argue that the experience gained on the first emulation project would improve the results of the second project. To avoid this criticism, a different person did each project. The argument, then, is that with two people the differences in education, experi-

ence, motivation, etc. affect the accuracy of the results and possibly affect the fairness of the comparison. One could also claim that the choice of a minicomputer for the experiment rather than a large-scale computer does not allow generalization of the results.* We can only respond that both microprogrammers are bright, capable people, that there are fewer and fewer distinctions between minicomputers and large-scale computers, and that we can only hope that the reader will appreciate our attempt to make a fair experiment. The test case was the emulation of the HP-2115 computer.

The Hewlett-Packard HP-2115 computer

The Hewlett-Packard HP-2115 is a predecessor of the HP-2100 and HP-21MX computers currently marketed by Hewlett-Packard. The HP-2115 has 70 basic instructions that operate on two accumulators, two flags, and (in this emulation) 8192 words of 16-bit main memory. This instruction set includes several special input-output instructions for up to 64 devices and provides for multi-level indirect and paged addressing. The HP-2115 will handle up to 64 different interrupts. Although the Hewlett-Packard manual lists 70 basic instructions, most of the opcodes in the shift-rotate group and alter-skip group can be combined to form more than 100 additional useful instructions.

HP-2115 emulation using traditional techniques

The emulation using traditional techniques was implemented by a graduate student in Spring 1975. This microprogram was written in TRANSLANG, the assembly language for the D-machine. He did not attempt to use the techniques of structured programming. He debugged the microprogram directly on the D-machine by running HP-2115 programs and examining the registers at each microstep. As we indicated above, this primitive approach is unfortunately typical of the way most microprograms are developed. This student had ideal conditions for using this approach in creating the emulation:

(1) He was thoroughly familiar with the HP-2115. He had spent two years as an M.I.T. undergraduate using this computer.
(2) He was thoroughly familiar with the D-machine. This student was in charge of maintaining the D-machine and the implementing enhancements to it.
(3) He was the sole user of the D-machine during that period. Usually a microprogrammer must compete with other users for access to the machine.
(4) The hardware was fixed and operational. With computer manufacturers the computer is usually being built and debugged while the microprogram is being debugged.† It is often difficult to distinguish between an error in the microprogram and an error in the hardware. With formal microprogram verification, the hardware is not needed until after the microprogram is verified, i.e., much later in the development cycle.
(5) There was software available to test his program. When a new instruction set is created, there is usually no program available to test the microprogram.

After he was satisfied that his microprogram worked, we tried running diagnostic programs for the HP-2115 on the emulation. Diagnostic programs are used to test if the hardware is working properly. This is a widely used technique to test accuracy of emulation. The argument is that if the diagnostics believed that the HP-2115 "hardware" was working correctly, the emulation of the HP-2115 must be working correctly. Running the diagnostics uncovered two errors in this emulation.

Subsequently we discovered that one of the unassigned opcodes performed a useful function. This opcode was obviously not tested by the diagnostics, but its use was apparently so widespread by users of the HP-2115 that they defined this opcode in a later programming manual. We were also surprised that the diagnostics did not test some apparently unusual combinations of options in the I/O instructions. The student left this out of his emulation because he doubted that it would ever be used. (In March 1976 he completed the emulation of all unusual I/O instruction options.) In writing the specification for the STRUM HP-2115 emulation, another error was discovered. Memory reference instructions can use the current page address to supply a portion of the operand address. When an interrupt occurs the program address register is not modified but the computer selects and executes an instruction from the first 64

TABLE I.—A Comparison of Size and Speed of Microassembly, Unoptimized Strum, and Optimized Strum Emulations of the HP-2115 Computer

	Microassembly	Unoptimized Strum	Optimized Strum
Number of Micro-instructions*	946	1124	940
Number of Nano-instructions*	162	123	161
Size in bits*	25504	25856	25344
Average number of cycles executed per instruction**	72.9	66.4	58.6

* The Burroughs D-machine has two levels of control memory so the size should be compared either in bits or number of microinstructions.
** The Burrough D-machine comes in several microcycle times from 50 to 500 nanoseconds.

* The selection of which computer to emulate was made by the person who made the emulation using traditional techniques, not by the author. It should also be mentioned that the language was designed before the computer was selected.

† A referee mentioned that usually a microprogram simulator is available while the hardware is under construction. Our experience is that the simulator and its corresponding hardware quickly diverge. While the simulator is useful this divergence combined with time constraints often force the microprogrammers to have access to the computer before the hardware is debugged.

locations in memory. It was not clear from Hewlett-Packard documentation whether a memory reference instruction that requested address modification from the current page would get the information from the program address register or from the current page of the instruction (page 0). This emulation used the program address register but a test of another Hewlett-Packard computer revealed that the address bits always came from page 0. These three examples show the danger in relying on diagnostic programs as the only method of verifying an emulation. This emulation used 946 microinstructions which required 162 unique nanoinstructions. The execution speed is shown in Table I. The values in these tables are given in terms of D-machine microcycles. The D-machine at UCLA runs at 800 nanoseconds per microcycle although Burroughs has developed versions that run as fast as 50 nanoseconds per microcycle.

Strum HP-2115 emulation

The second emulation of the HP-2115 using a high level language, structured programming and formal program verification was implemented in Spring 1976 by the author. We had serious reservations about this experiment, mainly because this microprogram would be the first microprogram that used the STRUM system. A simpler test case to catch any bugs in the STRUM compiler and program verification system was planned, but lack of time ruled this out. The result of the output from the optimized code is shown in Table I.

Testing the HP-2115 emulation

The testing of the STRUM HP-2115 emulation was performed on November 11, 1976 by the creator of the other emulation and the author. The test consisted of executing the HP-2115 diagnostic routines. These routines exhaustively test all opcodes and most register values for the memory reference, alter skip, shift rotate and overflow instructions. We did not know what to expect, because this was the first test of the STRUM compiler, and the first large-scale test of the STRUM verification system. The verification process uncovered 30 errors in the emulator divided equally between the executable microprogram, the specifications, and the assertions. The program and verification formulae were corrected by hand. Although one large (about 2000 lines of NUCLEUS code) program has been verified,[10] this program was never executed. We were in the uncomfortable position of using untested software tools to verify a program that would immediately be subjected to exhaustive testing.

Surprisingly no errors in the STRUM emulation were uncovered by the HP-2115 diagnostic programs. Four errors were found in the code generated by the compiler, but had we run a simple compiler test program beforehand, all these

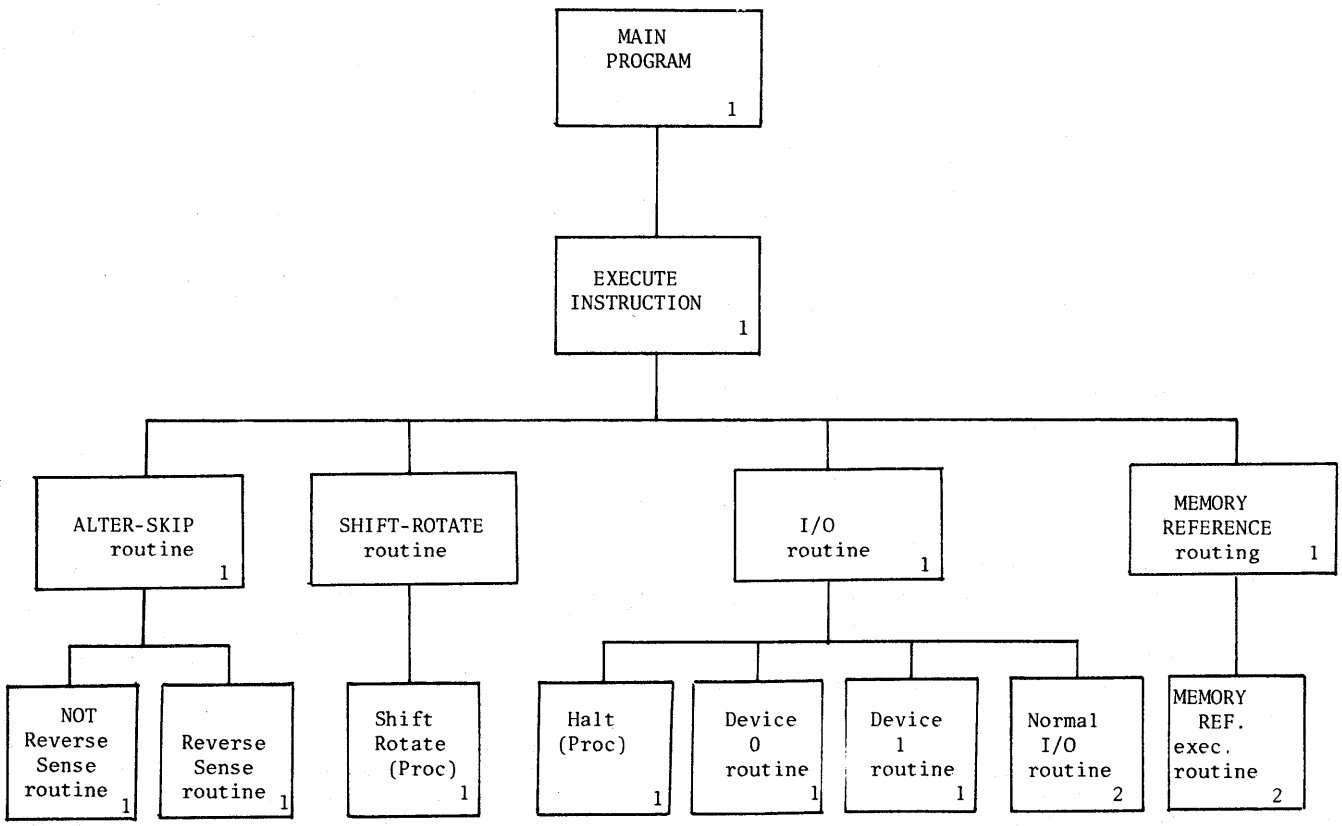


Figure 2—STRUM HP-2115 microprogram hierarchy (1 means one page in listing; 2 means 2 pages in listing)

errors would have been detected. This was indeed a gratifying result. To the best of our knowledge the STRUM emulator is the largest formally verified and tested program.

Summary and results

Although all the results were gratifying, we did not expect the STRUM version of the emulator to be faster than the TRANSLANG version. Many believe that high level language microprogramming is too inefficient to be useful.[3] Possibly an explanation for the unexpected result is the difference between "microprogramming-in-the-small" and "microprogramming-in-the-large." For small examples (10 to 20 microinstructions) it is feasible for a microprogrammer to spend hours optimizing microcode. However, with a 1000 instruction microprogram, it is extremely difficult to consistently convolute one's thinking to write efficient microcode. Also, high level languages and hierarchical structure make it easier to perform the global optimizations that may save more time or code than any instruction-by-instruction optimization. Perhaps the most exciting result was the success of the verification of microprogram. Although working with only a prototype system, we were able to verify a complete emulation of a real computer on a real microprogrammable computer. Moreover, although the exhaustive testing performed by the diagnostic programs did find four bugs in compiler, it did not reveal any errors in the STRUM emulation. Not only did the verification uncover program errors, it uncovered errors in the specification and assertions. No matter how practical testing may seem, it will not uncover errors in documentation (specification) nor in the comments (assertions). The microprogramming language STRUM proved to be very easy to use. The use of macros and good control structures resulted in a well-structured microprogram. Figure 2 shows the structure of the HP-2115 emulation. Each of the 14 boxes corresponds to a functional section of the emulator. Only two sections occupied more than one page of the listing and 12 of the 14 boxes are macros rather than procedures. STRUM object code and the original microassembly emulation contained many more jumps, labels and called statements than the STRUM source program (see Figure 3). The STRUM source program also contained fewer loops because the "handshaking" firmware used to access memory includes loops that are generated by the STRUM compiler.

HP-2115 EMULATIONS

	STRUM SOURCE	STRUM OBJECT CODE	ORIGINAL TRANSLANG EMULATION
LOOP'S	2	42	11
GO TO'S	0	144	144
LABELS	0	166	123
CALLS	16	37	104

Figure 3—Frequency of use of statements which make programs more complex

CONCLUSION

With respect to the microprogramming field we believe this experiment has demonstrated the validity of the use of each facet of the STRUM approach: use of a high-level microprogramming language, structured microprogramming, and formal microprogram verification. While we believe it is most effective to use all facets of the STRUM approach, the reader should be forewarned that the development of such system is a tremendous undertaking. The development of production versions of an efficient optimizing compiler and verification system is probably a five to ten man year effort. Hopefully tools will be created which will simplify this task.

Readers interested in a more detailed analysis should see References 4, 8, 11 and 12.

REFERENCES

1. Wilkes, M. V., "The Best Way to Design an Automatic Calculation Machine," *Manchester University Computer Inaugural Conference Proceedings*, p. 16.
2. Husson, S. S., Microprogramming: Principles and Practices, P-E, 1970.
3. Falk, H., "Hard-Soft Tradeoffs," *IEEE Spectrum*, Vol. 11, No. 2.
4. Patterson, D. A., "STRUM: Structured Microprogramming System for Correct Firmware," *IEEE Trans. on Computers*, Special Issue on Microprogramming, October 1976, pp. 974-985.
5. Dijkstra, E. W., "Notes on Structured Programming," EWD 249, Technical University, Eindhoven, Netherlands, April 1970.
6. Floyd, R. W., "Assigning Meanings to Programs," *Proceedings 1967 AMS Symposium on Computation.*
7. DeMillo, R. A., R. J. Lipton and A. J. Perlis, "Social Processes and Proofs of Theorems and Programs," *Fourth ACM Symposium on Principles of Programming Languages*, Los Angeles, January 1977.
8. Patterson, D. A., "Verification of Microprogram," Ph.D. Thesis, University of California, UCLA-ENG-7707, January 1977.
9. Davis, R. L. and S. Zucker, et al., *Aerospace Multiprocessor Final Report*, Technical Report APAL-TR-73-114, June 1973.
10. Ragland, L. C., "A Verified Program Verifier," Ph.D. Thesis, University of Texas, Austin, Texas, May 1973.
11. Patterson, D. A., "Analysis of an Experiment in Firmware Engineering," in preparation.
12. Patterson, D. A., "Formal Verification in an Unverified Environment: An experiment," *Proceedings of the Sixth Texas Conference on Computer Systems*, November, 1977, pp. 4A-13-4A-18.
13. Cottrell, J. R., "A Verification Condition Generator for a Structured Microprogramming Language," UCLA-ENG-7606, March 1976.
14. Patterson, D. A., "The Design of a System for the Synthesis of Correct Microprograms," *Proc. 8th Annual Workshop on Microprogramming*, September 21-23, 1975, pp. 13-17.
15. Tucker, A. E. and M. J. Flynn, "Dynamic Microprogramming: Processor Organization and Programming," *CACM*, Vol. 14, No. 4, April 1971, pp. 240-250.

Code optimization techniques for micro-code compilers

by C. J. TAN
IBM T. J. Watson Research Center
Computer Sciences Department
Yorktown Heights, N.Y.

INTRODUCTION

With the ever expanding volume of system functions directly implemented in microcode and the growth of microprocessor applications, it has become necessary to design high level language compilers for these machines to achieve high programming productivity. However, because of the very low level architecture of many of these machines, compilers that generate efficient code for these machines have not been produced.

An optimizing compiler, called PL/MP, has been designed which is capable of supporting a variety of microprocessors such as the Motorola M6800 as well as machines like the IBM 370 model 145, and several microprocessors designed and used internally at IBM.[1] PL/MP uses a set of novel, machine dependent, and the more conventional high level, machine independent optimization techniques to achieve high object code quality. Experimental results have indicated that the machine dependent optimizations play a very significant role in enhancing the quality of the object code produced by the compiler. The most effective of these optimizations are

1. Register allocation: optimized binding of compiler source and temporary variables to fast machine registers.
2. Low level addressing code optimization: code motion and common expression elimination for addressing code.
3. Code consolidation: combining a set of scattered simple instructions into a single more complex instruction supported by the object machine.

Basic to all of these optimization techniques are the algorithms for obtaining information concerning the use of variables throughout the program. These data flow analysis algorithms,[2,4,5] are used throughout the compiler for both machine independent and machine dependent optimizations. In this paper we will describe the code consolidation process, the environment in which it is carried out, and its dependence upon global data flow analysis. Algorithms for register allocation, code motion and hoisting have been published elsewhere,[2,3,6] and will not be discussed in this paper.

THE PL/MP COMPILER

The PL/MP compiler accepts a high level language such as PL/I and generates efficient object code for a micro-machine. The compiling process takes the source code through several levels of internal text as shown in Figure 1.[1] Note that the front end of the compiler is concerned with the machine independent transformations and optimizations, while the back end is concerned with the machine dependent transformations. The interface between them is the A-text. Operands in A-text are generic, symbolic registers of arbitrary length.

Frequently, micro-machines have a large number of fast registers. Even though these registers often have inhomogeneous usage constraints, it is important to optimize the use of these resources. Hence an inherent design philosophy in PL/MP is to assume, early in the front end, a machine with an indefinite set of registers. Source level variables are loaded into the register space and kept there. Their values as well as those for the compiler generated temporaries are stored into memory only if absolutely necessary. The traffic between memory and register space is thus kept to a minimum.

The internal A-text[7] is basically a register transfer language. However, it is general enough to support a large class of high level languages and low level micro-machines. It consists of primitive and complex operations, which may or may not be all supported by a given object machine.

Operands in A-text are constants, labels or generic registers. All computational operations are register-to-register. The only operations addressing memory space are loads and stores. Thus all addressing code has been exposed at this level. A sample list of primitive and complex A-text units is shown below.

Note that the complex A-text operations, such as add indirect, load indirect, etc., can all be replaced by a sequence of equivalent primitive operations. For most ma-

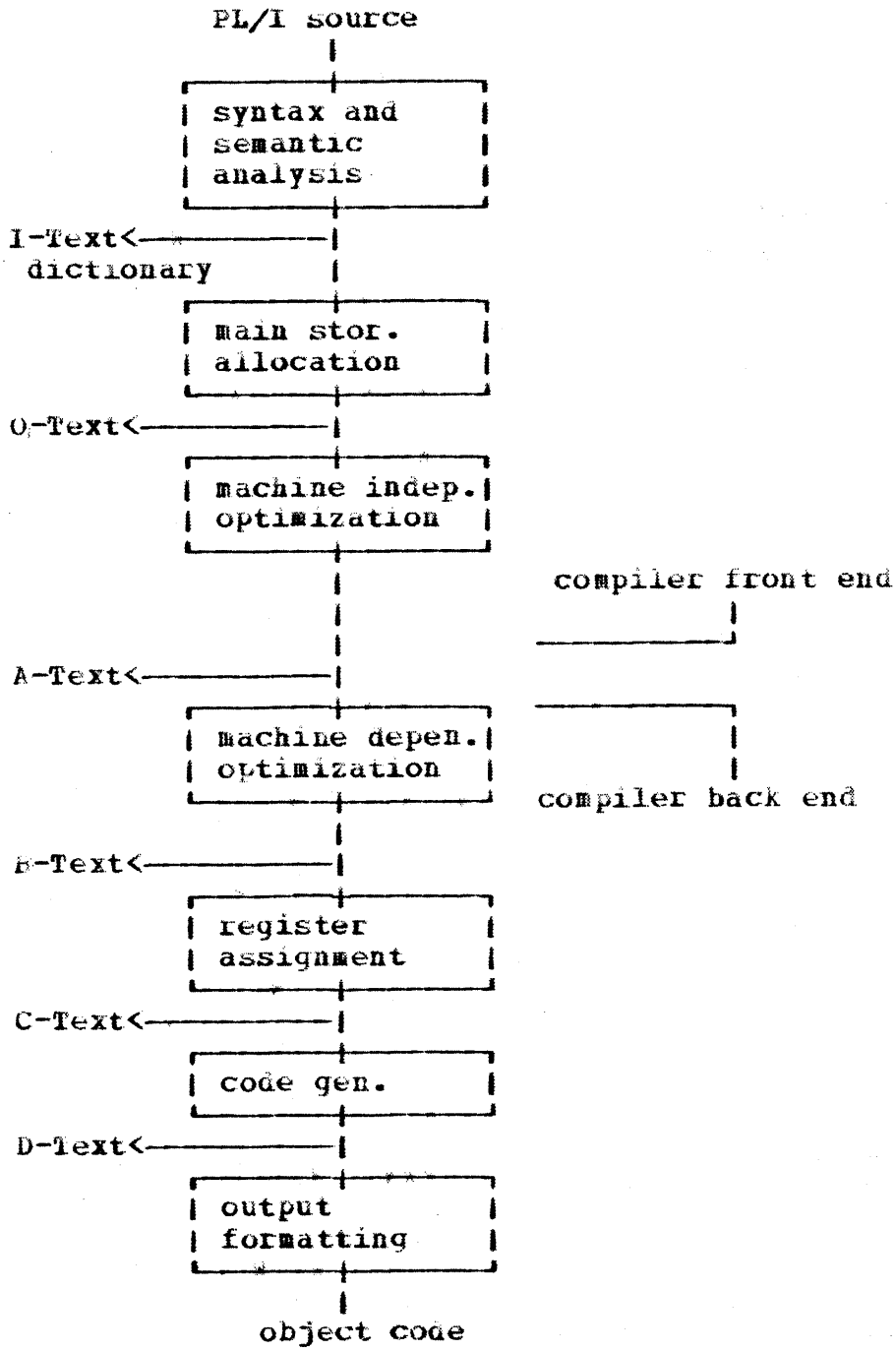


Figure 1—PL/MP structure

chines an add-indirect-with-offset, for instance, will be more cost effective than the equivalent set of primitive instructions. However, if the variables are used at several places in the program then under certain conditions many primitive addressing instructions can be optimized out by code motion or hoisting.[2]

THE ENVIRONMENT FOR CODE CONSOLIDATION

Assume that we have a machine with a set of general registers R_1 through R_{16} such that only R_1 through R_4 can be used as memory address registers. We further assume that the machine supports indirect addressing with and without

TABLE I.—Primitive and Complex A-text Operations

Operator	Operand	Comments
Primitive Operations		
ADD	R_1,R_2,R_3	add registers 2 and 3 and store result in 1
ADDC	R_1,R_2,R_3	add with carry
BLE	L,R_1,R_2	branch to L if R_1 less than or equal to R_2
LOAD	R_1,p	load into R_1 from location p
NOT	R_1,R_2	store into R_1 one's complement of R_2
CALL	L	call subroutine L
.		
.		
.		
Complex Operations		
LOADO	R_1,p,n	load with offset n[R_1<-MEM(p+n)]
LOADIO	R_1,R_2,n	load with offset indirect[R_1<-MEM((R_2)+n)]
LOADI	R_1,R_2	load indirect
ADDI	R_1,R_2,R_3	add indirect
ADDIM	R_1,p,k	add immediate[R_1<-p+constant k]
ADDIO	R_1,R_2,R_3,n	add indirect with offset

offsets. Thus, in this respect the target machine is at a higher semantic level than that of the primitive A-text.

Figure 2 shows a segment of an example in which we want to add a variable v to an array A of length 100 bytes. Both A and v are stored in a data area whose origin is at address p. The array's first element coincides with the area's origin and variable v is stored at an address 100 bytes beyond this origin. Thus, as shown in the figure, we first load into a register @a the address of array A. At program point p4 the array element is then loaded into register x through the load indirect instruction. The variable v is similarly loaded into register y, and it is added to x and stored in register z. Finally, the contents of z is stored back into the array, and the array element address @a is incremented by one to get to the next element. After iterating through the loop 100 times the program branches to point p7 where we proceed to other parts of the overall program. Figure 3 shows, in primitive A-text, the program after addressing code optimization by code commoning and moving the addressing code outside of the loop.

Notice that the sequence of code (p2,p3) which loads the variable v into the symbolic register y can be combined into the single instruction 'load indirect with offset', i.e.,

LOADIO y,@b,100 [R_y<-MEM((@b)+100)]

Once the above consolidation is performed the code sequence (p2,p3,p5) can be combined, so resulting, at p5, in a single add-indirect-with-offset instruction

ADDIO z,x,@b,100

Figure 4 shows the sample program after code consolidations at p2, p3, p5, and p7.

Notice that in the above consolidation process we have eliminated the register y containing the value of the variable v. Hence these consolidations are effective only because the symbolic register y, itself, is not referenced elsewhere in the program. Consider a slightly different situation, shown in Figure 5, where the register y is used at point p8. The subtract immediate instruction involves the register y and cannot be consolidated with the addressing code for y. This implies that we must keep the value of v in this registery.

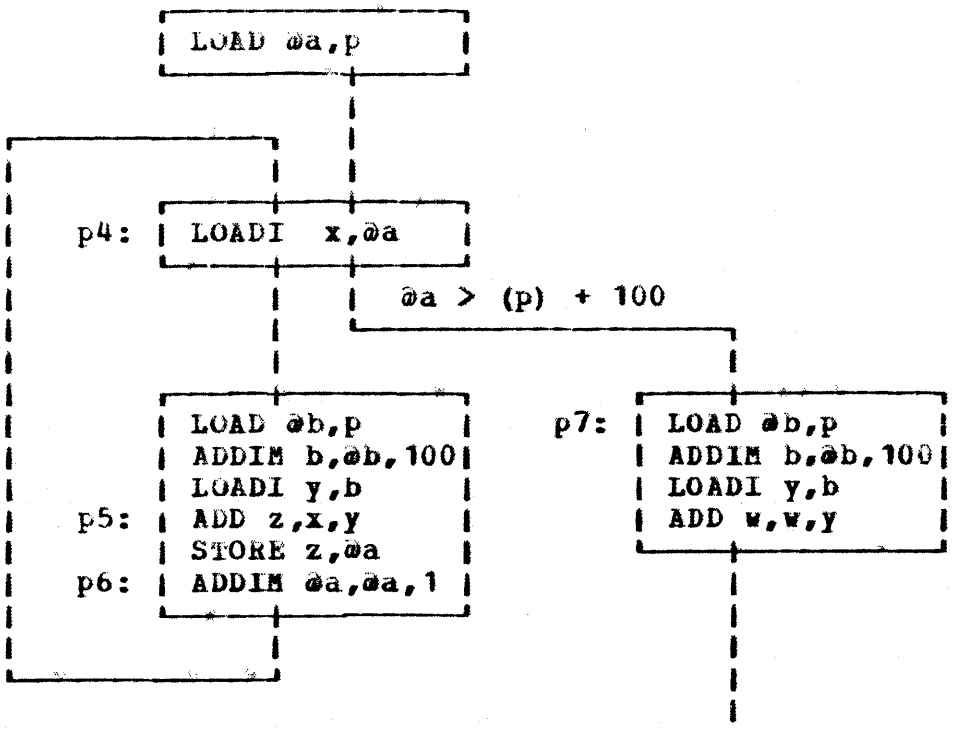


Figure 2—Sample program before addressing code optimization

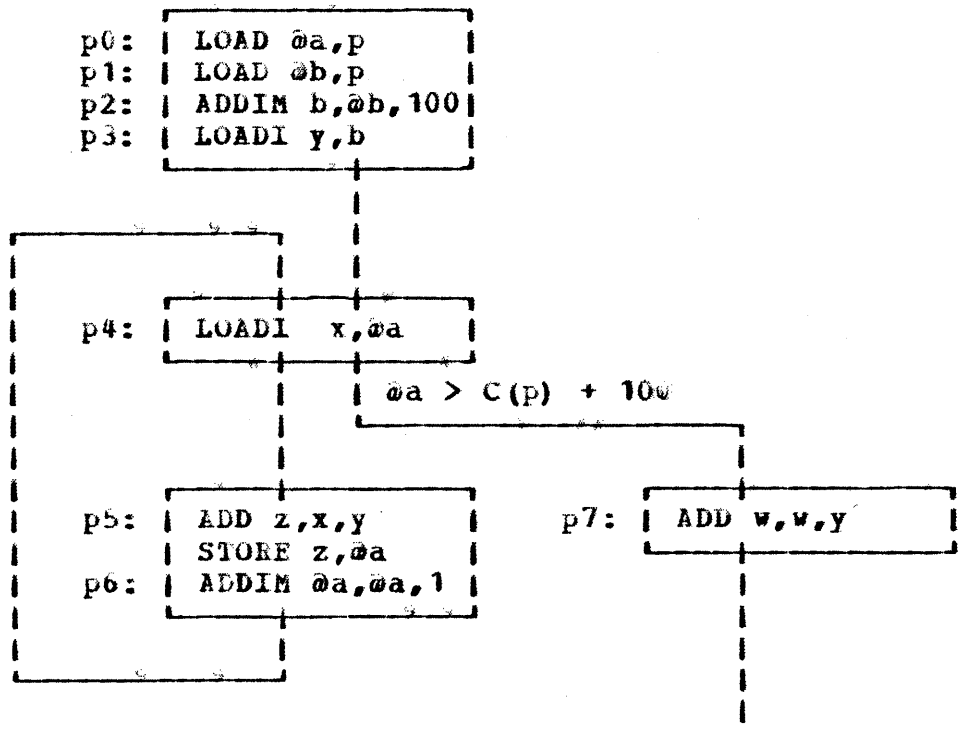


Figure 3—Sample program in A-text

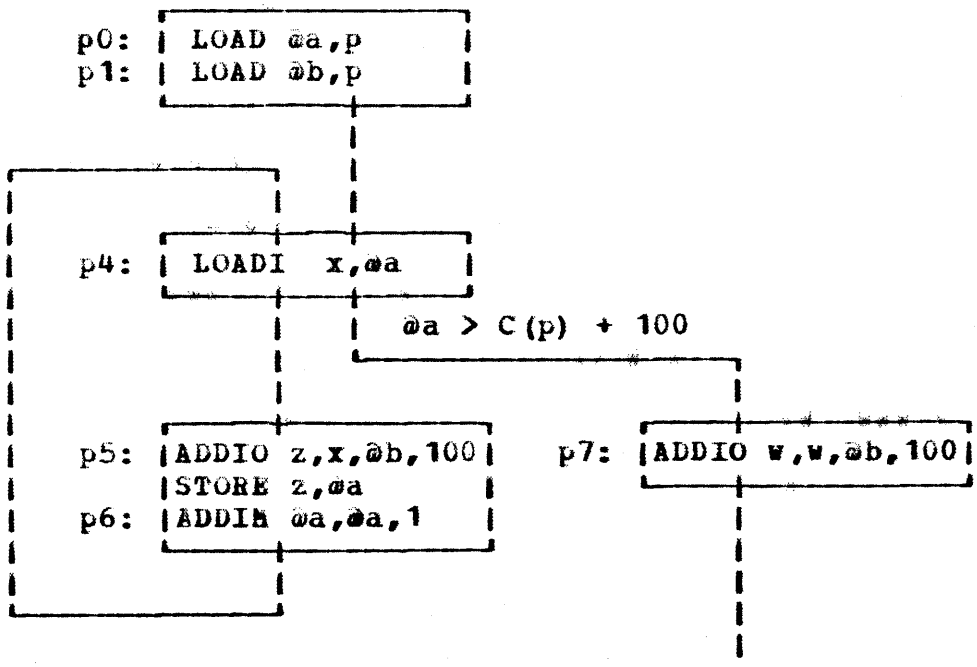


Figure 4—Sample program after code consolidation

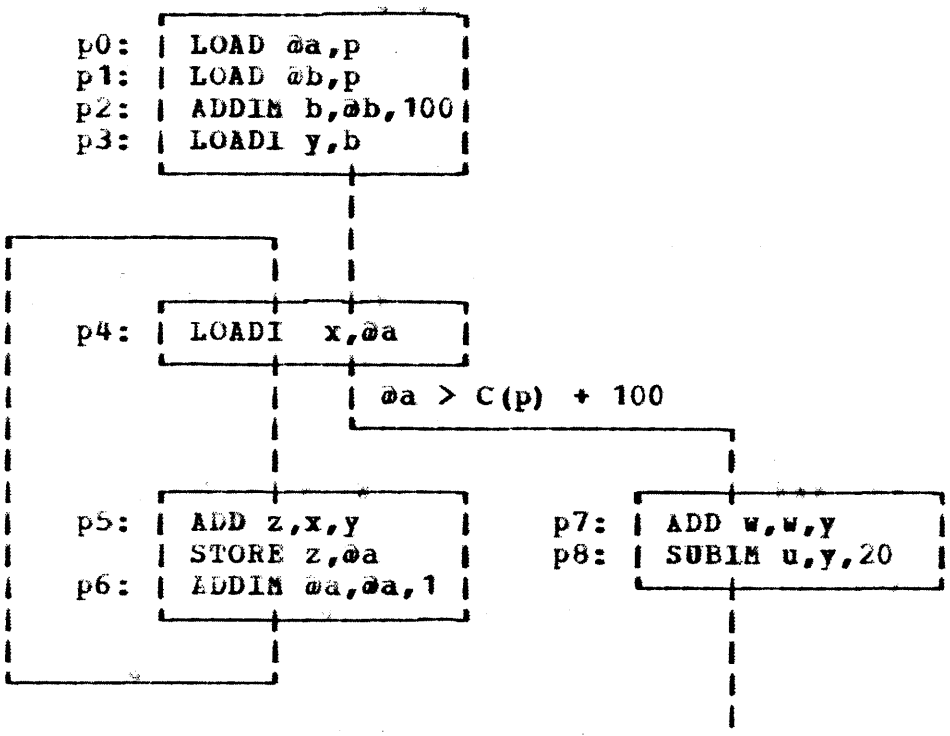


Figure 5—Sample program in A-text

Therefore, the instruction first consolidated from the sequence at (p2,p3), i.e.,

LOADIO y,b,100

cannot be deleted in the next consolidation iteration. Hence, for this case, replacing the code at p5 and p7 with the more complex instructions would actually increase the object code size.

It is clear from the above example that the decision to perform code consolidation can be made intelligently only after examining the global usages of the variables in question. It is not surprising, therefore, that the compiler strategy is to first generate A-text with only primitive instructions. The primitive A-text, with addressing code fully exposed, is then subjected to global code optimizations such as commoning and code motion. Finally, the possibility of code consolidation is explored to generate an equivalent A-text using complex instructions where advantageous.

CODE CONSOLIDATION ALGORITHM

We have seen, in the previous section, instances where scattered code sequences can be consolidated. We will now describe the procedure for performing code consolidations. The procedure involves: (1) recognizing a possible consolidation candidate, and (2) performing the actual consolidation if it is profitable to do so.

Code pattern classification

For many representative machines, the primitive A-text code patterns that correspond to complex instructions consist of an addressing code sequence followed by a computational operator, which we will call the terminal operator. Furthermore, it is often possible to find a set of terminal operators that share an identical addressing code sequence. Sequences having this commonality are grouped together to form a pattern class. For instance, the code sequence p2, p3 in Figure 3 is a member of the class that contains indirect-with-offset operations such as LOADIO, STOREIO, and so forth. Similarly, the sequence at p3, p5 is a member of the indirect operations class.

For many machines there exists a partial ordering among the code pattern classes. For example the code sequence (p2,p3,p5) corresponds to the complex operation ADDIO. Notice that the sequence (p2,p3) forms the complex operation LOADIO, and similarly (p3,p5) forms the operation ADDI. Hence the arith-indirect-with-offset(AIO) class is greater than the class containing LOADIO as well as the class containing ADDI.

The patterns for our fictitious machine may be classified as below, where the first instruction in the sequence is called the 'token', and the primitive instructions are divided into arithmetic, memory-register classes, and so forth.

From this classification we can derive the substitution rules for these patterns as shown in TABLE III.

TABLE II.—Pattern Classes

Class	Instructions	Token	Terminal code
C1	⟨mem-reg-ind-off⟩= {LOADIO,..}	ADDIM	⟨mem-reg-ind⟩= {LOADI,STOREI,..}
C2	⟨arith-ind⟩= {ADDI,SUBI,..}	LOADI	⟨arithmetic⟩= {ADD,SUB,..}
C3	⟨arith-ind-off⟩= {ADDIO,...}	LOADIO ADDIM	⟨arithmetic⟩= ⟨C2⟩
.			
.			
.			

That is, the instruction in C3 (e.g., ADDIO) replaces either the sequence consisting of LOADIO followed by the corresponding arithmetic instruction (e.g., ADD) or the sequence of ADDIM and a sequence from C2 (e.g., ADDI). Notice that all tokens are specific instructions either primitive or complex. Furthermore, a code sequence in a secondary pattern class, such as C3, can always be formed from sequences of a smaller class. It is obvious that if no code pattern in C1 or C2 can be profitably consolidated, for instance, then no pattern in C3 can be profitably consolidated either. Hence it suffices to perform consolidation starting from the smallest element in the partial ordering defined by the pattern classes.

Code sequence recognition

Before we can make a judicious consolidation decision based on relative costs, for a given A-text and a specific pattern class C, we must first find a valid code sequence in that class. This involves (1) finding a sequence of code satisfying the substitution rules, and (2) checking that the operands in this sequence satisfy certain data flow constraints.

Notice that, starting from the token of a code sequence, each operation defines a new symbolic register. This defined value, unaltered, must be used in the immediate successor operation in the sequence. Furthermore, the target variable at the token, called the substitution variable, must not be altered along the path containing the code sequence so that it can be substituted at the terminal operation.

Since the elements of a sequence may be scattered throughout the program, it is a difficult task to check for these conditions. Fortunately, very efficient global data flow techniques have been developed[5] that enable us to generate certain global relations among program variables. We will now introduce some of these data flow concepts, and then show how they can be used to find a valid code sequence for a given class.

TABLE III—Substitution Rules for Pattern Classes

⟨C3⟩::=LOADIO⟨Arith⟩|ADDIM⟨C2⟩
⟨C2⟩::=LOADI⟨Arith⟩
⟨C1⟩::=ADDIM⟨Mem-reg-ind⟩

Let (v,p) denote a variable v defined at program point p. We say (v,p) *reaches* point q if there is a path from p to q such that v is not redefined along that path. If q is a program point with two or more predecessor statements then we say (v,p) is *available* at q if (v,p) reaches q through every one of its predecessor statements. Availability implies that if v is also defined at another point, say r, then (v,r) cannot also *reach* q. Since the reach and availability information can be determined efficiently for any program, we can state the following

Variable Flow Condition: if p_i and p_j are two consecutive code points in a code sequence then the variable defined at p_i, as well as the substitution variable for the sequence, must be available at p_j.

We have already mentioned that every substitution sequence starts with a known token. Hence having found a token for a given class we must start scanning the text to find the next code point in the sequence that satisfies the variable flow condition. The following constraints allow us to search through the program rather efficiently. First we will introduce two additional data flow concepts.

The first concept we need is the so called *variable liveness*. If a variable v defined at p reaches a point q and is such that there is a subsequent use of (v,p) in a path starting at q then we say (v,p) is *live* at q, otherwise we say (v,p) is *dead* at q. Thus we need not search along a path where the variable being searched is either dead or not available.

The second concept we need is that of *node dominance*. (For the following discussions we use the term node to denote a block of straight line code in a program. Hence p0,p1,p2,p3 in Figure 5 constitute a node. Similarly, p5,p6 is a node.) We say a node d in a program flow graph *dominates* another node n if every path from the program entry node to node n goes through node d. For a given program a dominator tree can be obtained such that there is an edge connecting nodes d and n if and only if d dominates n. For the program in Figure 3 the dominator tree is simply the graph minus the loop edge from p6 to p4.

Notice that if $(p_1,p_2,p_3, \ldots ,p_i)$ is a substitution sequence then in order to replace this sequence at point p_i by a more complex instruction, the node containing p_1 must dominate that containing p_2, and so forth. Therefore, starting from the token at p_1 we need only follow the path defined by the dominator tree. Fortunately, for any flow graph a dominator tree can be obtained very efficiently as a part of the general data flow analysis.[2,5]

The overall algorithm

Once we have found a valid substitution sequence the next question is whether or not it is advantageous to actually consolidate the code. Notice that if $(p_1, p_2, \ldots, p_l)$ is a substitution sequence, then we can always replace the operation at p_l by the complex operation that is logically equivalent to the function of the sequence. However, since the replacement operation is usually more complex than the original operation at p_l, we presume it is advantageous to consolidate only if all the code at the other points in the sequence can be deleted. This is possible only if the intermediate variables are all used by terminal operations of the same class and found to be valid for consolidation. Otherwise, we cannot delete them, and hence there may be no advantage in replacing the original operation at p_l by a more complex one.

Starting with the token we search for all possible substitution sequences of the given class. If a code sequence is found to violate either the variable flow condition, or one of the intermediate variables is used in a statement not allowed by the sequence substitution rules, then the intermediate addressing code cannot be deleted. Hence no code consolidation will take place and the process terminates.

Note that if a code pattern is greater than another pattern in the pattern class ordering, as is the case with C3>C2 and C3>C1 in our previous example, then in searching for a sequence for C3 we must encounter a consolidated text unit in either class C1 or C2. Otherwise, since we always start the consolidation process for the smaller class first, this implies that no code can be advantageously consolidated for the patterns in C1 and C2. Hence no code pattern in class C3 can be consolidated either. In fact, the substitution rules for C3, as shown in Table III, clearly indicate that a valid sequence in that class must contain either a C1 or a C2 operation.

The above discussion can be summarized by the following:

Algorithm C

Input: Program expressed in A-text, program flow graph, variable liveness, availability, and dominator tree, pattern classes and substitution roles.

Output: Consolidated A-text.

Method:

1. Starting from the smallest pattern class C that has not been previously processed, perform the following steps. If no unprocessed class is left then terminate.
2. If consolidation failed for all lesser classes of C then go to step 1, otherwise proceed to step 3.
3. Starting from the root of the dominator tree, look for a previously unprocessed token of the given class. If a token is found then proceed to step 4, otherwise go to step 1.
4. Search for substitution sequences in class C along the paths defined by the dominator tree.
5. If no substitution sequence is found or if the variable flow condition is violated then return to step 1.
6. If a use of an intermediate variable is made by an operation not part of a code sequence then go to step 1.
7. Replace terminal operations found by appropriate operations in class C, and delete the intermediate addressing code. Return to step 3.

CONCLUSIONS

We have discussed some of the techniques used for machine dependent code optimizations for a micro-program compiler. One of these techniques, namely code consolidation, has been discussed in detail. It should be obvious that the compile-time efficiency of these optimizations is heavily dependent upon the efficiency of the global data flow analysis package used by the compiler.

The code consolidation process, as described, may seem to be rather complex. However, for typical real machines, the number of pattern classes turns out to be rather small, and the substitution sequence length is also usually short. Hence with the use of a high-powered data flow package, the algorithm indeed can produce very efficient code with reasonable computation.

Experiments conducted for one of the commercially available micro-processors, namely the M6800, have shown that code consolidation together with addressing code optimization have contributed a 30-40 percent improvement of the object code produced by the PL/MP compiler.

ACKNOWLEDGMENT

It is a great pleasure to thank Hans Schlaeppi, manager of the PL/MP compiler development group, for providing the environment under which the research reported in this paper was carried out. Thanks are also due to many of my colleagues in the compiler group, without whose assistance this work would have been impossible.

REFERENCES

1. Davies, K., J. Kim, J. Rutledge, H. Schlaeppi, C. J. Tan, R. D. Villani, A. B. Wadia, and H. Warren, PL/MP Workbook, May 1977.
2. Aho, A. V. and J. D. Ullman, *Principles of compiler design,* Addison-Wesley, 1977.
3. Allen, F. E., "Program Optimization," *Annual Review of Automatic Programming,* 5, 1969.
4. Allen, F. E. and J. A. Cocke, "Program Data Flow Analysis Procedure," *CACM,* 1976, pp. 137-147.
5. Rutledge, J. D., A Preliminary Overview of the FDS Data Flow Package, (Internal PL/MP memo), May, 1977.
6. Tan, C. J., Register Assignment Algorithms for Optimizing Compilers, IBM research report RC6481, April 1977.
7. Ris, F., The Syntax and Semantics of A-text. (Internal PL/MP memo), May 1976.

Microprogram verification considered necessary

by W. C. CARTER, W. H. JOYNER, JR. and D. BRAND
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

Microprograms have been around for some time now, and so have errors in them. Therefore techniques have been developed for finding the errors. We review these existing techniques and explain why we are working on a new one. The main advantage of our method is its ability not only to detect errors, but (if there are no errors) to give a formal proof that the microprogram does perform according to specifications. Using our approach we are working on actual machines and actual microcode. We discuss our experience with such proofs of correctness and what kinds of errors we are finding. The reader will also find a comparison of problems in microprogramming with problems of general software reliability.

WHY IS MICROPROGRAM VERIFICATION CONSIDERED NECESSARY?

"Microprogramming Trend Considered Dangerous" is the title of a letter in the ACM Forum.[1] The letter continues, "Thus the development and maintenance process for microcode will increasingly resemble that of systems and applications programs. And therein lies mortal danger." This paper will briefly trace the history of microprogramming and microprogramming support systems. As microprogramming has developed, better support systems have evolved so that microprogramming has been a useful tool. Now, with the widespread development of microprocessors on a chip, and use of read/write control stores, new ideas for support systems must be developed. This paper then describes one such idea, symbolic simulation for microprogram verification, and indicates its necessity. The goal of this symbolic simulation of microprograms is to find all errors in the microcode (and correct them) while proving that the final code is correct. To do this, the Language for Symbolic Simulation (LSS) is used to prepare an APL-like executable description of the architecture (or program) specifications. The same language is used to prepare a description of the microprocessor. Then it is proved mathematically that the microprocessor when controlled by the actual microcode simulates the specifications, using symbols for data instead of bits. This language and procedures are described, together with an automated set of program aids, the Microprogram Certification System (MCS). Some results are given of applying these techniques and MCS to real programs and computers. These results indicate that microprogram verification by symbolic simulation is feasible, and micro code maintenance problems indicate that verification is necessary.

WHAT ARE PREVIOUS TECHNIQUES?

When Wilkes first proposed microprogramming in 1953 and built the EDSAC II,[2] computers were built entirely of discrete components. The store holding the microprogram was also built of discrete components and the microprogram was debugged like the rest of the computer—by graduate students. When hardware components were available so that microprograms became commercially feasible (in the late 1950's), read-only storage was preferred because of its speed and low cost.[3,4] Fixing errors in the microprogram then became slow and awkward, so special simulators were written to help debugging. Unfortunately every data flow or control flow change necessitated changes in both the simulation program and the microcode. A race developed—would the simulator or the computer be running first?

However, computers controlled by microprograms have many convenient features. An architecture for a family of computers is desirable, but needs a complex instruction set for the top of the line and for ease of programming.[5] A rich instruction set costs very little more than a spartan set, if the computer is microprogrammed.[6] Rewriting application programs for new machines is expensive. But emulation of the previous computer instruction set by a new computer using the facilities of microprogramming allows the old program to run.[7] Maintenance of many distinct computers in a family is difficult, since the hardware for each computer is different. Again, using a microprogram allowed special maintenance techniques—microdiagnostics, scan-in, scan-out, log-out, instruction retry, etc. to be designed.[8]

This microprogram complexity doomed the individually written simulator, and more complex design automation systems were written. One, the Controls Automation System,[9] was written for the S/360 design effort. The computer hardware at the register transfer level was described by a Fortran-like programming language, including bit string handling. Design changes meant only a recompilation to produce an input for the simulator. The microprogram was written

at an assembly level, but with mnemonics for the microinstructions chosen by the design engineers. The assembled input to the simulator was kept in a program file, in which any single subroutine could be changed. The event driven simulator could be instructed to set up desired starting and ending for the simulation, to select the desired microcode for simulation, to produce snapshots of the registers at variable breakpoints, to dump in emergencies, to trace with or without timing checking, etc. This and similar systems[10] were a definite advance, but problems remained.

WHAT WAS WRONG WITH THEM?

Another litany on the difficulties of program testing in the middle 1960's is hardly needed. Anyone who has written a program to be used extensively by other people knows the problems. "Testing detects the presence of bugs, but never their absence"[11] is the well-known refrain. "Enough" test cases can never be generated in our limited time on earth. How is a good selection to be made? The coverage depends upon the skill and dedication of the individual who wrote the microprogram. But even if all paths are traversed by test cases, faults which are data dependent can still remain. Moreover, simulation in any guise is slow and expensive.

Thus the systems did reach the field with faults in their control. Diagnosis of the computers themselves was difficult. Most faults occurred many levels deep—a particular combination of instructions and data turned them into observable errors. The best routines for exposing these faults were the higher level language compilers. (The diagnostics run, but the compiler doesn't.) Therefore capturing the environment so the fault could be discovered and fixed was difficult. A knowledge of many different design levels became necessary—from circuits (were race conditions triggered?) to logic design (did that decoder work correctly?) to microcode (was that transfer necessary?) to machine code (what series of instructions was traversed?) to the compiler (what statement is being compiled?). The well-known technique of trying a fix to see how it worked could not be applied to microcode in read only storage.

WHAT HAPPENED?

Human ingenuity and perseverance triumphed again. Microprogramming was—and is—a great success. The computers worked well, and the new facilities made available more answers per dollar spent.

Hardware test support equipment[12] came into use—expensive but necessary. Special computers, equipped with read/write control stores were built. Facilities for rapidly changing the control store contents were added, and fixes were performed in minutes. Changes could now be tried rapidly, and if they seemed to work all system and test programs could be run against the microprogram change before final commitment. Also, automatic data test generators were built, and millions of pseudo-random data patterns could be processed.

Now the bandwagon effect began and is still continuing. SIGMICRO (ACM) and the Technical Committee on Microprogramming (IEEE CS) were formed and began to hold their yearly workshops, presenting new ideas. If microprogramming is good for central processors it must be good for I/O equipment.[13] Look at all the new functions which can be performed! Need reliability/availability? Add special progressive scan test routines.[14] Need help with communications? Provide independent computing power.[15] Does the operating system take too much time and space? Put part of it in microcode.[16,17] Would you like a slightly different computer, or direct execution of higher level languages? Just provide the correct microcode.[18] Does your terminal take too much of your CPU time? Provide an intelligent terminal. All of these advances were fostered by improving hardware technology which provided more and more logic facilities in less and less space. Now (in the 1970's) we have various types of microprocessors on a chip—CPU's of various types,[19] I/O processors,[20] standard busses, and everyone (almost) can build his own distributed system.

Help to do this is provided in abundance. Do you think that the use of higher level languages (Pascal, Algol, Fortran, etc.) will help your programming problems? Just provide the correct environment and the standard language can be used. Do you think that structured programming will help? Courses are readily available. Are you using your system for control? You can buy hardware test equipment which will simulate the environment in which your micro will run.[12] Many advances in program testing are now available. Do you want a system which will analyze your program, and produce test inputs so that all paths will be traced at least once, and all instructions executed (if possible)? Such systems are available.[21] Would you like to learn more about selecting test cases, devising adequate test procedures, applying support systems to your own problems? Help is available.

So what's the problem? Once again in our enthusiasm we are building faster and faster—and reliability is falling behind. Letters to the editor such as Lehman's "Microprogramming Trend Considered Harmful"[1] point out the dangers in this. All of the techniques listed so far test specific cases—and the untested cases can still be erroneous. Microprograms are hidden from all but the designer. Should we explain to the checker at the local supermarket that the buttons on the cash register send data to a computer which then uses this information to get a "correct" answer? Most of the time everything works correctly—but when errors occur rectifying them is expensive—and fixing the system is even more expensive. (Have you bought a very cheap pocket calculator recently?)

WHAT ELSE IS AVAILABLE?

The problems that microcode poses are also posed by ordinary programming systems when such systems are written to be used extensively by people distinct from the originators. Ten years ago a formal, mathematically oriented approach, called program verification, was formulated.[22,23] In this approach a series of assertions, written in first order

predicate calculus, are provided for the program input, output, and various intermediate points in the program. In particular every loop must satisfy an assertion called a "loop invariant" specified by the user. Devising these invariant assertions is not easy, even though many papers have been written, each proposing a procedure which is effective in various special cases.[24] Various simple program modules (written in high level languages) have been proved correct, and more informal proofs have been given for larger examples.[25] However, as with all proofs, the proofs are only as good as the hypotheses and the number of people who have actively checked them.[26] A difficulty with the pure form of this standard approach is that by starting from an externally given program text it necessarily loses sight of the program's genetic origins. This deficiency has been recognized;[27,28] and proposals made to cure it. In general the idea has been to recreate, or use, a formal expression of the generating steps by which programmers construct programs in the first place. These proposals are still in the research stage.

Another approach has been studied by King[29] and Clarke[30] and experimental program aids have been constructed. They use as a major part of program validation the execution of the parts of the program between assertions using symbols, not the actual data values. The difficulty here (which is one of the difficulties in automatic program verification also) is defining and implementing the execution semantics for the symbols and the operators used in the programming language.

Symbolic execution can be seen to have several deficiencies. Symbol manipulation is costly and slow. Again the question of coverage arises. The data paths are represented symbolically, but which paths can be covered? Mostly main lines? What is the total cost? Will all errors be found? In Howden's experiments[31] neither ordinary good testing techniques nor symbolic execution found all known errors in a Fortran program. In addition there are the real world problems of the actual computer representation of data, with overflow, interrupts, etc. Another difficulty is that symbolic execution also ignores the knowledge that the programmer has about the specifications, and it does not check the specifications. Recent studies have shown that wrong or inadequate specifications account for 50 percent of program errors, and these errors are the most expensive to find.[32]

WHY DO THESE TECHNIQUES HAVE PROMISE FOR VERIFYING MICROCODE?

The techniques of program verification have been applied for the most part to higher level language programs and have met with limited success. Fully automatic methods are almost nonexistent—often a fairly detailed human analysis of the data structures and control structures used is necessary before automated aids can be applied. Also, such proof of necessity deals with some abstraction of what actually happens in a computer when the compiled code produced from these programs is executed. Errors can therefore result because the actual running program is not the program which was verified. Why should program verification techniques have any more promise when applied to microprograms, especially to actual microcoded implementations, than they have in other applications?

As is described in the next section, our approach requires a complete formal description of the computer on which the microcode is to be run, and the actual microcode—the array of bit patterns—is used with a symbolic interpreter. Though this makes necessary a more complex description, because overflow, invalid array indexing, etc. must all be explicitly described, errors which would go undetected if these factors were ignored can be detected.

The use of a single data structure—a two-dimensional array of bits—simplifies both the symbolic execution and the specification of the Floyd-type assertions or their counterparts. The registers, memory, I/O channels, switches, etc. of the computer are all represented as arrays. Also, the operations on these arrays are limited to storing and retrieving strings of bits addressed by indices, concatenation, reshaping to other dimensions, and certain arithmetic and logical operations. Problems with program verification often arise when more complicated data structures are involved; if there are such structures in a microprogram, they are explicitly implemented in terms of the more primitive notions, and verification takes place over that more restricted domain.

The fact that microprogrammers are not constrained to write in a "structured" way (and often do not in order to increase time or space efficiency) probably increases the difficulty of formulating and placing correctness assertions in microcode. However, once that is done, the control structure of a machine description which interprets microcode is simple, and symbolic execution can proceed in a straightforward manner.

The way in which we use symbolic execution to prove microcode correct is another factor which improves our chances for success. As described in the next section, we do not place assertions at various points in the microcode which relate values of machine components to previous values, or describe a state (e.g., sorted, permuted, etc.) which the computer must satisfy. Rather, we equate the machine components with components of a higher level description, a description of the architectural specifications which the microcode is supposed to implement. In this way we can partially avoid the necessity for quantifiers, single-program inductive invariants, and references to initial or previous values of machine components.[33]

The most compelling argument for the feasibility of microprogram certification is our experience. We have designed and implemented an automated system to aid in verifying microprograms, and have used it to detect and correct errors in "real" code.

WHAT ARE THE DIFFERENCES BETWEEN VERIFYING MICROCODE AND VERIFYING HIGHER LEVEL PROGRAMS?

A microprogram is just a special kind of program. The main difference between firmware and software is in the semantics of the programming language and the emphasis on the speed of execution for firmware. Microcode is inter-

preted by hardware, whereas higher level programs can easily have their semantics defined in terms of another program—an interpreter. This makes a difference because a verifier must be given the semantics of the programming language. Thus for verifying microcode it is necessary to describe a piece of hardware as an interpreter. An interpreter for a higher level programming language presents an idealized view of the environment (this is the whole idea of higher level languages). The lower we go towards the physical realities the more we have to face to them; here are some of the realities we are faced with.

Hardware has the inherent capability of operating in parallel. Parallelism in software is certainly very common, but software can use the cushions provided by the lower layers of software. For example, memory interlocks prevent two processes from accessing a shared variable at the same time. Nobody will provide such interlocks for hardware, where shared variables are physical wires; therefore we have to watch for the possibility of races. Cooperating processes in software can take advantage of synchronizing primitives, which shield the software designer from the idiosyncrasies of hardware. No such shield is available for a microprogram, which is the shield for others. The microcode itself must handle such things as I/O, interrupts because of external causes, program-caused overflows, and other exceptions, which are "hidden" from the high level language programmer and which most program verification techniques ignore.

At the specification level, there are parts of the description which are left intentionally ambiguous, so that different models of a simple architecture (such as the 360 series) may implement them in different ways. In verifying a particular implementation, we must handle this ambiguity in such a way that it does not "lock in" the specification to a particular implementation technique.

On the other hand there are aspects of microprogramming, which make verification easier. Microprograms operate on one fixed data structure—bit vectors. Further, these operations are not very sophisticated, in the sense that the bit vectors are not supposed to represent notions very different from themselves (at least as far as the architecture is concerned)—a bit vector typically represents a number. Thus the statement of correctness is usually close to what the microcode does, but the results are expressed in terms of numbers rather than bits.

All these characteristics of microprograms make them good subjects of mechanical verification. They are so concerned with small details that it is difficult for people to examine them carefully. At the same time they are not sufficiently intellectually challenging for a person. But these are exactly the characteristics which machines find easy to handle, and therefore computers have a better chance in competition with people when verifying microcode than when verifying higher level programs.

HOW SHOULD MICROCODE BE VERIFIED?

Testing techniques are based on the ability to recognize an erroneous procedure. Verification proves that a program written at a level above machine code is correct. What is needed is a system which will find all errors in a given program as it runs on a computer in a reasonable length of time. To do this three capabilities are necessary: formal specifications which are natural and easy to produce; coverage determination, i.e., the ability to recognize a correct procedure; and the ability to execute the code as existing in the microprocessor so that analysis of the results will determine the fault.

MS	$2^{24}\times 8$
GPR	16×32
ACC	1×33

Figure 1

To formalize specifications we model all facets of system operation using the APL-like Language for Symbolic Simulation (LSS).[34] This description is an abstract machine whose state consists of a facility vector describing the machine (or storage) components and a control structure which permits parallel asynchronous operation. The facility vector components, two-dimensional arrays of bits, are described by dimension statements, as in Figure 1. These facilities are an 8 bit wide Main Storage with 2^{24} locations, sixteen 32 bit General Purpose Registers and a 33 bit Accumulator. Operations upon the facility vector are specified by a library of macro routines written using APL-like operators, as in Figure 2.

This macro models a S/370 add register instruction. The 2's complement addition is modeled by decoding ($\perp$) the contents of the registers from bit strings to integers, adding the integers, then encoding ($\top$) the result in 33 bits ($(33\rho 2)$). This value is assigned (:=) to a. SETCOND is called, which sets the S/370 condition code. The low order 32 bits of a (bits 1 to 32, 0 origin) are put in GPR[R1;].

The macro SETCOND is shown in Figure 3. SETCOND shows the conditional structure of the LSS macros. OVERFLO is a predicate from the predicate library. If it is true, then bits 34 and 35 of the Program Status Word are both set to 1. Otherwise, if the result is zero, the condition code is $\langle 0\ 0\rangle$, or positive $\langle 1\ 0\rangle$, or finally negative $\langle 0\ 1\rangle$.

The last example is OVERFLO, which shows some of the logic operations available. Here, z is the 32-bit result of adding the 32-bit quantities X and Y (Figure 4).

The LSS language is carefully defined in Reference 34. Only two kinds of objects are necessary for LSS, integers and two dimensional matrices of bits (truth values are one bit matrices).

Modeling computing systems by describing their state vectors and the state transformations is intuitively plausible. LSS specifications are executable, i.e. given a state vector

```
ADDREG (R1, R2)=
      a:=(33,ρ2)⊤(2⊥GPR[R1;])+2⊥GPR[R2;]
      SETCOND (a,GPR[R1;],GPR[R2;])
      GPR[R1;]:=a[1+ι32]
```

Figure 2

```
SETCOND(X,Y,Z)=
    OVERFLO(X,Y,Z)→PSW[34+ι2]:=⟨1 1⟩
    ELSE          →X[1]=0→
                      (X=32ρ0)→PSW[34+ι2]:=⟨0 0⟩
                      ELSE→PSW[34+ι2]:=⟨1 0⟩
                  →X[1]=1→PSW[34+ι2]:=⟨0 1⟩
```

Figure 3

value as input, a symbolic output state vector value may be obtained by symbolic execution. A symbolic value is an expression built from the operators of the LSS language and symbols representing the initial values of the variables, after appropriate simplification. For example, if A has the value ⟨11110000⟩ and B has a symbolic value $B, then the assignment statement A:=A&B becomes A:=⟨11110000⟩&$B; A is given the new value $B[4],⟨0000⟩ (the "," here is catenation). The adequacy of the system model may be tested by executing cases of special interest. The model acts as a special purpose machine, i.e., it carries out one set of specifications.

LSS is also used to describe the hardware attributes of the computer on which the specified architecture or program is to be implemented by code. This implementation description acts like a general purpose computer. When the assembled code to be verified (a matrix of bits) is inserted in memory, and a set of symbolic inputs is given, symbolic execution and simplification will determine the outputs.

To determine if the desired action is identical to the action of the code, we use symbolic simulation. Heuristically speaking, the execution of the machine description and actual code simulate the execution of the specifications if everything the specifications can do the machine and code can do, though possibly in a different way. Because the two facility vectors (specification and implementation) may have different components, to carry out the simulation it is first necessary to specify a set of relations between them. These simulation relation components specify, for selected pairs of specification and implementation control points, relations between the state vector quantities which must hold at these points. See Reference 35 for a complete and formal description.

To carry out the verification a symbolic execution tree, or proof tree, must be constructed.[36] An automated, interactive system, MCS (Microprogram Certification System), has been designed and implemented to aid in this verification (see Figure 5). A node, or goal, in the tree represents a class of states of the system at one point in time. In general, a node will consist of

(a) two state vectors—an assignment of symbolic values to facility variables;
(b) two control points—positions on the control trees;
(c) a predicate list—a list of conditions that the initial values must satisfy in order that the states represented by this node can be reached.

```
OVERFLO(X,Y,Z)=(X[0]≠X[1])&(Y[0]=Z[0])
```

Figure 4

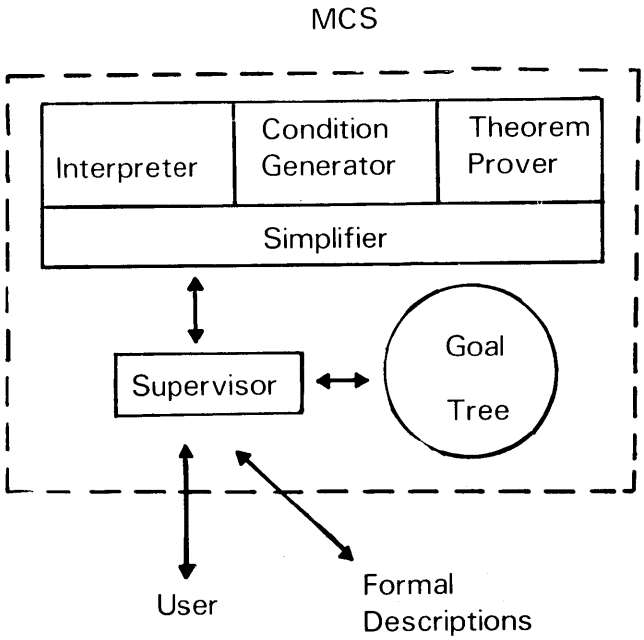


Figure 5—Parts of the microprogram certification system

The root of the tree (which represents the entire problem of proving the simulation) has a son, called a simulation relation node, for each of the simulation relation components. The state vector, control, and predicate lists of these nodes correspond to points at which the specified relations must hold. The values of the elements of the facility vectors in these simulation relation nodes satisfy the relationships necessary so that the implemented machine, under the control of the program, will simulate the specification machine. A branch in the proof tree represents a computation path. The leaves of the proof tree represent all possible final states.

The following process is used to build the proof tree. First, the simulation relation nodes are generated. In these nodes the most general symbolic values such that the conditions of the corresponding simulation relation component hold are given to the variables of the state vector. Any conditions which cannot be asserted in this way are placed on the predicate list. The next step is to run each abstract machine, performing symbolic computation until another corresponding pair of control points in the simulation relation is reached.

One machine is arbitrarily chosen to be run first. At this initial point variables have as values either symbolic constants (symbols representing unknown but fixed values) or values determined from the simulation relation. Control proceeds as in normal execution. When symbolic constants are encountered in expressions being computed in assignment statements, the value assigned is a simplified combination involving operators and symbolic constants.

When symbolic constants occur in predicates evaluated to determine possible branches, a single flow of control may not be able to be determined. The predicate may evaluate to a Boolean expression involving symbolic constants, such

as "$X=0," which cannot be evaluated to "true" or "false." In such a case all possible logically independent results of evaluating the predicate must be considered; in the previous case $X=0 and $X≠0. The program doing the symbolic evaluation will generate a node (subgoal) for each independent result, add a predicate expressing the truth of the result to the predicate list, and simplify the result. In the case being considered, two subgoals will be generated. For $X=0, 0 will be substituted everywhere for $X; for $X≠0, this predicate will be put on the predicate list. The user then chooses the path he wishes to follow (the program is interactive) and begins again. The remaining paths will be traversed later. If the predicate involving symbolic constants evaluates to true or false, no path branching occurs.

After generating a series of nodes and branches, the control corresponding to a simulation relation component is reached. Then the other abstract machine is run until a simulation control point for that level is reached.

Now the system verifies that the pair of control points reached defines the control component of a simulation relation node. The values of the two facility vectors are substituted into the simulation conditions of this component, and a goal is generated for each simulation condition. Using the predicate list, a proof is attempted for each simulation condition. The process is repeated for each node of the tree which is not a final node. In the process a complete goal tree is formed.

Newly created expressions are simplified at once to prevent the propagation of unsimplified forms. At present MCS has over 400 rules for the simplification of APL and logical expressions. If a theorem simplifies to "true," then the goal is achieved (and theorems are the only goals generated in proving simulation that can be achieved directly, without generation of subgoals). If the theorem cannot be proved by the theorem prover, its simplified form is the theorem of the pattern in a single generated subgoal so that analysis by the user is facilitated. It may occur that a theorem cannot be proved, or that a pair of stopping points not corresponding to a simulation relation component is reached, or that a stopping point is not reached. Then an error must be sought in the code or in one of the descriptions. In addition the occurrence of an unexplained branch will signal the presence of an error.[37]

In MCS, theorem proving, which is simplification of APL-like expressions, occurs at a number of places and forms an important part of the system.

This general process becomes more complicated when the asynchronous parallel operation of the system must be considered. Our method of modelling parallelism involves considering the processes as composed of K≥2 separate subprocesses running sequentially, but interleaving arbitrarily. No notion of time is built in. How the subprocesses interleave is immaterial as long as no shared variables are used, since these subprocesses are and remain independent. When shared variables are used, we have to consider which subprocesses can be executed between two references made by a single subprocess to a shared variable. We also must consider how the shared variables and the subprocesses interrelate in the process execution. To model this we provide LSS operators to express when interleaving may occur; when interleaving is not specified a subprocess runs without interruption.

A solution has been proposed by Brand and Joyner to consider the K separate subprocesses as each running with its own control tree and to add two statements to LSS, DELAY and WAIT, so that the shared variables can be correctly handled during the arbitrary interleaving of the subprocesses.[38]

EXPERIENCE USING SYMBOLIC SIMULATION

The MCS is now being applied to a version of a real computer, the NASA Standard Spaceborn Computer-2 (NSSC-2). This version is called the Hybrid Technology Computer (HTC). This version was produced at the IBM Federal Systems Division at Huntsville, Alabama.[39] Its architectural specifications require it to support the 86 System/360 Standard Instructions (no decimal or floating point operations). It has the usual sixteen 32 bit general purpose registers and 16 to 64K bytes of storage (with an extended protection mechanism). A single I/O channel supports three types of I/O: device initiated I/O, CPU initiated I/O and Direct Memory Access I/O. Device initiated I/O has two classes. Buffered I/O allows a device to transfer data to/from a table in main memory without knowing the location or size of the table. External interrupt permits a device to interrupt the normal program sequence. CPU initiated I/O also provides two types of services: command and 16 bit data transfers to or from I/O devices or CPU to I/O 16 bit control operations. Direct Memory Access is transparent to the NSSC-2 microprogram controlled hardware unless a hardware error occurs, when the microprogram is called. Each use of the I/O interface, whether CPU or device initiated, follows a certain protocol, depending on the I/O command.

Describing the NSSC-2 CPU was straightforward. The state vector facilities are essentially those listed in the S/360 programming manual. The basic CPU macros are all of the type (and approximate length) of those described earlier in this paper. This control, however, involves recursive calls, so that instructions will be fetched and executed indefinitely, or until some interrupt or other external signal intervenes. The basic idea is shown in Figure 6. The macro which handles the interaction between the CPU and the I/O is much more complicated because of the asynchronous parallel action of the CPU and I/O devices. This is carefully described in Reference 40. The description takes 25 double spaced pages of computer printout.

This architecture is implemented by a data flow that has a 16 bit wide data path, three working 16 bit registers, a 16

```
cpu=
    (SOFT-STOP=0)→DO-INSTRUCTION
                  SERVICE-INTERRUPT
                  CPU
    (SOFT-STOP=1) → Ω
```

Figure 6

bit wide store whose storage data register does double duty, a 64×16 bit scratchpad memory, and a 16 bit I/O register.[39] The data flow is controlled by a 1K read only memory of 64 bit wide horizontal micro-instructions, another 64×16 read only memory for computer instruction decoding, a 32 bit instruction register, and various sized registers and signals for speed of operation. The facility vector consists of the storage data flow registers and control signals. The macros describing the action of the 21 types of CPU oriented microcommands are again straightforward. The I/O protocols depend basically upon 2 other microcommands. These protocols require that certain acknowledgments be sent by the CPU, that the CPU wait until certain signals are received from devices, that interrupts not be ignored indefinitely, etc. Verification of these microprogram controlled protocols is an important part of microprogram certification (see Reference 40 for a complete description). This description takes 20 pages of computer printout.

One part of the HTC microcode, where the new treatment of I/O was heavily used, is system reset. This is code executed upon start of the HTC, and its job is mainly to initialize various data structures. It also initializes the I/O channel status; here it is important to ensure that the protocols are strictly observed. During system reset the microcode uses a certain piece of code that is used also for other purposes. For these other purposes the request line (which can be raised by the I/O interface) is sensed. The operation of system reset, however, depends on this line being down. To ensure this the computer must prevent any device from being serviced by the I/O interface. Therefore before entering the shared piece of code the microcode resets the interface. (That is, service of any device will be terminated). After resetting the interface the proper operation of reset is checked using this shared code. To prevent a device from getting service during this checking, the command line (under CPU control) is enabled. If these precautions were not taken, system reset would not be performed correctly when a device happens to request service at exactly the wrong moment. Needless to say, we could hardly expect this kind of error to be detected by testing.

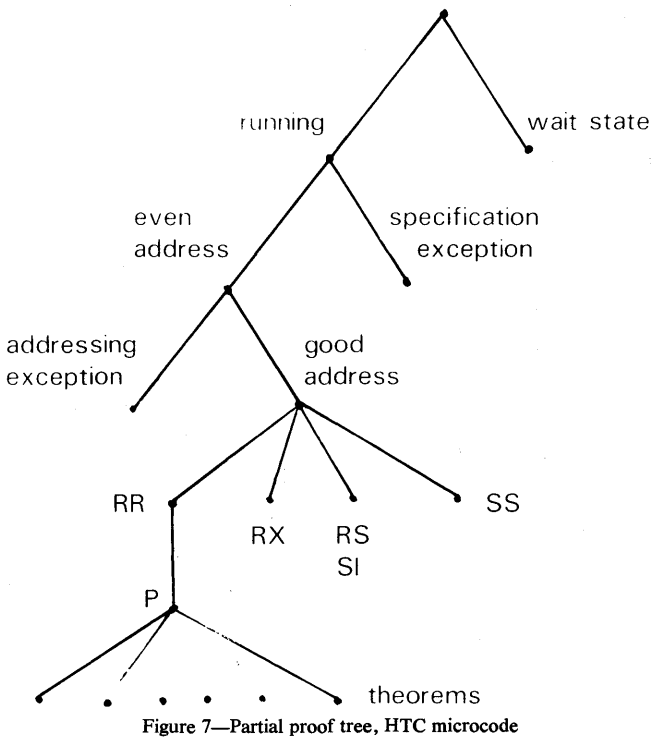


Figure 7—Partial proof tree, HTC microcode

In the course of trying to prove correctness of system reset we found that, after checking that the reset was valid, the microcode jumped to a routine which implemented part of the protocol for sending the storage data register information to an I/O device. This is not only a violation of specifications, but the information transmitted is random—whatever happens to be in the storage data register when the power is turned on. Moveover, in this routine the microcode sent an acknowledgment to the I/O interface without checking that the correct I/O routine had begun. If an unknown device were requesting service, it would receive the service acknowledgment even though no device input has been read. The device—and its input—would be actually unknown. After consulting with the author of the microcode it was determined that he used this feature for debugging purposes (indicating to Test Support Equipment that reset was finished) and had left it in the final product by error.

After eliminating this jump to the routine which violated the architectural specifications, we were able to verify system reset, following 5 paths and proving 28 theorems.

The next part of the microcode to be tested concerned the CPU IFETCH, 15 microinstructions, 3 paths and 45 theorems. Figure 7 gives an outline of the proof tree developed for this HTC microcode. In this part of the code we found that if a program overflow had been signaled, but upon return to the beginning of IFETCH an I/O interrupt was waiting, then the overflow would be ignored. We also found that no half-word instruction may reside in the last two bytes of memory. Since the memory contents are not available immediately, the first thing the IFETCH code does is fetch a word (16 bits) from memory, then immediately read the next word so 32 bits are read to fill the instruction register. If the instruction is 16 bits there is no slowdown, while if the instruction is 32 bits there is an increase in instruction speed. Unfortunately if the first 16 bits read are a 16 bit instruction in the last two bytes of memory, IFETCH tries to read the next two bytes, a memory overcapacity interrupt is given and IFETCH is never completed. This is an example of a bending of specifications, not a microprogram fault. A note in the programming manual would be better than slowing the HTC down.

Another 97 microinstructions were executed during symbolic execution of 10 S/360 RR instructions. Thirty-two paths were followed, and 480 theorems proved. In the Branch and Link instruction (BALR) the link address was stored before the branch address was accessed. Thus BALR R_i R_i became a NO-OP instead of a branch and link. This was not detected by being unable to prove a theorem; instead, it was detected by generation of an unexpected branch of the goal tree. (The error had been found by further testing

before we found it.) One other specification exception was found: bit 31 in the key word in the Set Storage Keys (SSK) instruction must be zero. This is counter to the instruction description, which says that bit 31 is ignored.

CONCLUSIONS

Some architectures, data flows, programs and microprograms have been described using LSS. Microcode has been symbolically simulated and errors found. Modelling systems and programs using LSS has been relatively straightforward, except when trying to describe very efficiently asynchronous parallel actions. MCS has run with reasonable speed during these verification runs, and it appears that MCS can be used to verify microprograms of moderate size. Thus, the approach of symbolic simulation does offer some hope for correct microcode before customer shipment, and automated methods have been successful in detecting errors in "real" microprograms, which would be very difficult or impossible to detect with program testing. Certification of microcode and machine level code may thus be the place where automated program verification techniques will reveal their applicability to actual software and firmware.

REFERENCES

1. Lehman, M. M., "Microprogramming Trend Considered Dangerous," *Comm. ACM,* 18, 6, June 1975, pp. 358-360.
2. Wilkes, M. V. and J. V. Ehringer, "Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer," *Proc. Camb. Phil. Soc. 49, part 2,* April 1953, pp. 230-238.
3. Eachus, J. J., "Logical Organization of the Honeywell H-290," *AIEE Trans. Comm. & Elec.,* Part 1, Apr. 1960.
4. Kampe, T. W., "The Design of a General-Purpose Microprogram-Controlled Computer with Elementary Structure," *IRE Tran. Elec. Comp. EC-9,* June 1960, pp. 208-263.
5. Amdahl, G. M., G. A. Blaauw, and F. P. Brooks, Jr., "Architecture of the IBM System/360," *IBM J. R&D,* 8, April 1964, pp. 87-115.
6. Brown, J. L., et al., "IBM System/360 Engineering," *Proc. FJCC,* Oct. 1964.
7. Tucker, S. G., "Microprogram Control for the System/360," *IBM Syst. J. 6,* 4, Oct. 1967, pp. 222-241.
8. Carter, W. C. et al., "Design of Serviceability Features for the IBM System/360," *IBM J. R&D,* 8, Apr. 1964, pp. 115-127.
9. Buckingham, B. R. S., W. C. Carter, W. R. Crawford, and G. A. Nowell, "The Controls Automation System," *Sixth Symp. on Switching Theory & Logic Des.,* Oct. 1965.
10. Zucker, M. S., "LOCS: An EDP Machine Logic and Control Simulator," *IEEE TEC, EC-14,* 6, June 1965, pp. 403-416.
11. Dijkstra, E. W., *NATO Conference on Software Engineering,* Conference report, Jan. 1969.
12. Kline, Barbara, "The Microcomputer Development System: An Answer to Microcomputer Design Needs," *Digest Compcon '77,* March 1977, pp. 152-154.
13. Chu, Y., *Microprogramming and Computer Organization,* Prentice-Hall, Englewood Cliffs, 1972.
14. Preiss, R. J., Chapter in: Breuer, M. A. *Design Automation of Digital Systems,* Prentice, Englewood Cliffs, 1972.
15. Infotech Information, Infotech State of the Art Report 23, Microprogramming and Systems Architectures, 1975.
16. Forbes, Brian et al., "Realizing a virtual machine," *9th Microprogramming Workshop,* Sept. 1976, pp. 42-46.
17. Brown, George, et al., "Operating System Enhancement via Firmware," *Digest 10th Microprogramming Workshop,* Oct. 1977, pp. 119-133.
18. Wilner, W. T., "Design of the Burroughs B1700," *AFIPS Conference Proceedings,* Vol. 43, 1973, pp. 489-498.
19. Session 15, "Mini and Multi Processors," *Digest Compcon '77, Micros, Minis and Maxis,* Sept. 1977, pp. 213-231.
20. Session 23, "Smart Peripherals," *Digest Compcon '77, Micros, Minis and Maxis,* Sept. 1977, pp. 301-315.
21. Jessup, W. H. et al., "ATLAS—An Automated Software Testing System," *Proc. 2nd Int. Conf. on Software Eng.,* Oct. 1976, pp. 629-636.
22. Floyd, R. W., "Assigning Meaning to Programs," *Proc. Symp. Appl. Math,* 19, 1967, pp. 19-32.
23. Hoare, C. A. R., An Axiomatic Basis for Computer Programming, *CACM 12,* 10, Oct. 1969, pp. 576-580.
24. German, S. M. and B. Wegbreit, "A Synthesizer of Induction Assertions," *IEEE Trans. Soft. Eng. SE-1,* March 1975, pp. 68-76.
25. London, R. L., "Experience with Inductive Assertions for Proving Programs Correct," *Symp. on Sem. of Algo. Lan.,* Lecture Note in Math #188, Springer-Verlag, pp. 236-252.
26. Gerhart, S. L. and L. Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodologies," *IEEE Trans. Soft. Eng. SE-2,* 3, Sept. 1976, pp. 195-208.
27. Burstall, R. M., "Program Proving as Hand Simulation with a Little Induction," *IFIP 74,* Stockholm, Aug. 1974.
28. Neuman, P. G. et al., "Software Development and Proofs of Multi-Level Security," *Proc. 2nd Int. Conf. on Software Eng.,* Oct. 1976, pp. 421-429.
29. King, J. C., "A New Approach to Program Testing," *Proc. Int. Conf. on Reliable Software,* Apr. 1975, pp. 473-481.
30. Clarke, L. A., "A System to Generate Test Data and Symbolically Execute Programs," *IEEE Trans. Soft. Eng. SE-2,* 3, Sept. 1976, pp. 215-223.
31. Howden, W. E., "Reliability of the Path Analysis Testing of Strategy," *IEEE Trans. Soft. Eng. SE-2,* 3, Sept. 1976, pp. 208-215.
32. Boehm, B. W., R. K. McClean, and D. B. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *Proc. Int. Conf. on Reliable Software,* April 1975.
33. Manna, Z., "The Correctness of Programs," *J. Comp. & Sys. Sci. 3,* 2, May 1969, pp. 119-127.
34. Carter, W. C., H. A. ElLozy, W. H. Joyner, and G. B. Leeman, "Techniques for Microprogram Validation," *Agardograph No. 224,* "Integrity in Electronic Flight Control Systems," April 1977.
35. Leeman, G. B., "Some Problems in Certifying Microprograms," *IEEE TEC C-24,* 5, May 1975.
36. Birman, A. and W. H. Joyner, "A Problem-Reduction Approach to Proving Simulation between Programs," *IEEE Trans. Soft. Eng. SE-2,* 2, June 1976.
37. Joyner, W. H., W. C. Carter, and G. B. Leeman, "Automated Proofs of Microprogram Correctness," *Proc. 9th Microprogramming Workshop,* 1976, pp. 51-55.
38. Brand, D. and W. H. Joyner, Verification of Protocols using Symbolic Execution," *Proc. Symp. Computer Network Protocols,* Feb. 1978.
39. Technical description of SUMC Computer Model 2B, *IBM Doc. 76W00013,* Dec. 1975.
40. Carter, W. C., W. H. Joyner, D. Brand, H. A. ElLozy, and J. L. Wolf, "An Improved System to Verify Assembled Programs," *IBM Research Lab. Report,* November 1977.

A panel session—Formal methods in programming—When will they be practical?

SESSION CHAIRMAN—KARL N. LEVITT
SRI International

Panel Members

Donald I. Good—University of Texas
Ellis Horowitz—University of Southern California
Barbara Liskov—Massachusetts Institute of Technology
Lawrence Robinson—SRI International

PANEL OVERVIEW—Karl N. Levitt

Formal methods, i.e., use of mathematical rigor, have been employed by research computer scientists in their attempt to develop general results for many aspects of computer science, e.g., computational complexity, undecidability, numerical analysis, programming language semantics. Much of this work has had little impact on those charged with producing working software systems. However, in recent years numerous researchers have suggested that by applying formal methods to the realization of systems, the quality of such systems could be significantly improved. Such formal methods could be applied in the structuring, specification, verification, and analysis of performance for systems. The position statements below explore the use of these techniques in the production of systems. The general opinion is that formal methods will ultimately assume a vital role, but for the present their use will be restricted to particular systems produced by skilled individuals. The use of the formal methods will gradually increase as the techniques are refined and applied to a larger variety of systems, as tools are developed to support their use, and as the general community becomes better educated in formal methods.

FORMALISM CAN HELP YOU—Lawrence Robinson

There is much resistance on the part of the programming community to the use of formalism in programming methods. However, there is also a problem in achieving reliable and maintainable software that meets its requirements. Despite the resistance, the application of formalism to programming techniques shows much promise in attaining a software product of high quality. At present, some advanced techniques are ready for limited use.

First it is useful to state exactly what formalism is. *Formalism* is a means for making logical and consistent arguments about a particular area. To formalize an area requires a set of definitions and axioms to characterize the area, and a set of rules of inference to validate or disprove logical arguments. Formalism can be used to understand, analyze, and reformulate both a problem and its proposed solution, and to verify that the proposed solution actually solves the problem at hand.

Other engineering disciplines make use of formalism to some extent (e.g., Laplace transforms in electrical engineering and partial differential equations in structural engineering). In fact, some aspects of software already make use of formal techniques, such as the exact specification of the syntax of a programming language. However, software engineering, because it is still a very young field, lags behind other engineering endeavors in the use of formal techniques.

In addition, the special problems inherent in software engineering require an even greater use of formalism than do the problems of other engineering endeavors. Two factors are responsible for software engineering's greater need for formalism:

- The complexity of large software systems exceeds that of other engineering systems by several orders of magnitude. Any engineering practices that are developed for software must place considerable emphasis on the management of this complexity. Current software engineering methods have not adquately done so. Formal mathematics has been the classical tool for managing complexity, by combining many cases into canonical groups, and by structuring complex concepts by the use of abstraction and formal definition.
- Software has a requirement for total exactness that is unique among engineering disciplines. Whereas the products in other fields are required to fall within continuous tolerances, the smallest error in a piece of software (a single bit) will sometimes result in the catastrophic failure of a system, rather than in degraded performance. Software engineering techniques must be able to guarantee the exact fulfillment of requirements. Formal techniques provide the only possibility of exactly stating the requirements of a piece of software, and for mathematically proving that the software meets its requirements.

Because software is so difficult an area, its development requires a great deal of attention to the analysis and planning that precede the coding. Current techniques underemphasize these early aspects of the software development process. Some new techniques require formal analyses of requirements and formal design specifications before coding can proceed. These techniques are ready for use by advanced systems programmers. A methodology developed at SRI International (HDM—the SRI Hierarchical Development Methodology) has been used for formally specifying the designs of several operating systems and many intermediate-sized systems; some of the systems have been implemented.

However, these techniques are not currently ready for everyone, and considerable training is required in order to use them. The major obstacle, however, is a resistance to change. But the education process (at universities, for example) is making these techniques available to people entering the software field. Gradually, programming will become closer to mathematics—more of a science than a black art. The changing process will be painful and fraught with resistance, but it hopefully will result in better and better software products.

SOFTWARE DESIGN BY ALGEBRAIC SPECIFICATION—Ellis Horowtiz

I'd like to adopt for the moment an extreme point of view so I can describe the range of application of algebraic specification to software design. A recent advance in software construction has been the recognition that one should begin by supplying a formal specification of the intended task. This notion applies both to programming systems as well as to individual programs. The debate now centers on the way such a specification will be phrased, and as with programming languages practitioners are developing strong biases for and against certain approaches. My own bias centers on algebraic specification.

Algebraic specification is not appropriate for specifying all possible tasks. For example, the technique does not seem natural for describing concurrency. What it does seem ideally suited for is the description of data types (data structures). Many computer applications can be conceived of as the transformation of data from one form into another. Then, the programming task becomes one in which the logical data structures must be precisely defined and then represented via other data structures which are available on a computer. The essential ingredient which makes the algebraic approach so successful for data type specification is the fact that a data type is composed of a set of related operations. These operations create, build-up or make smaller instances of the data type, they may decompose it into its constituent parts or answer questions about its form and content. All of these operations are best viewed together and algebraic specification naturally exposes these relationships.

The relationships between operations of a data type are given via a set of equations, usually called algebraic axioms. These equations show the effect that the operations have on instances of the data type. One important virtue of these equations is that they imply no representation. Thus the meaning of the operations of a data type is defined abstractly (i.e., without regard to the eventual representation), and the design process is broken into two logical phases: data type definition and implementation. My own experience leads me to conclude that an algebraic axiomatization is about as easy to compose for a data type as it is to describe the type in English.

The ease of specification might alone be sufficient to recommend algebraic axioms. But it turns out that there are other payoffs when one adheres to this software design scheme. In particular the testing and verification of the resulting software are all facilitated when one continues to design and implement all data types via the algebraic discipline. A software "environment" can be created which aids the programmer as he builds his system. At the USC/Information Sciences Institute several of us (D. Musser, J. Guttag) have created the Data Type Verification System (DTVS) which supports program creation by algebraic specification.

We begin by algebraically axiomatizing a single data type. Is our axiomatization correct? Before attempting a proof some testing would be desirable. DTVS will automatically synthesize an implementation from the axioms. This will permit testing at the level of the axiomatization without the trouble of implementation. Naturally such testing will be costly in terms of execution time, but experience indicates that only small cases are sufficient to turn up most errors.

When testing is completed, a representation for the data type in terms of other data types is devised. Then implementations of the operations in terms of the operations of the implementing data types are given. Once again testing is repeated, but this time DTVS will use the implementations as the semantics of the data type. Once we are satisfied, a proof of the correctness of the implementation is carried out semi-automatically by DTVS.

Specification is a process we continue to explore. It is possible to build a complex software system by strictly adhering to algebraic axioms. By doing so one gains both the virtues of a formal approach and the testing and verification environment. This offers much more than an ad hoc strategy. Finally, this approach is not inconsistent with the eventual use of a conventional programming language.

PRACTICAL BENEFITS OF RESEARCH IN PROGRAMMING METHODOLOGY—Barbara Liskov

In the past few years, considerable progress has been made in the area of programming methodology. Although all research in this area is interrelated, two main research directions can be distinguished. One direction is the study of software system structure, in particular study of desirable kinds of modules and module interconnections. The other direction is the study of the process of developing correct software having such desirable structure.

Study of software system structure has been particularly effective within the framework of research on programming

languages, especially the languages CLU[1] and Alphard.[2] This language work has succeeded in identifying new kinds of modules, and has provided precise rules governing the implementation and use of such modules. Each kind of module supports a kind of abstraction found to be useful in constructing software. The most important new kind of module is that supporting a *data abstraction*; however, there are other kinds of modules, and in addition rules governing the interaction of modules. These latter rules constrain and reduce the interconnections among modules.

Earlier work in structured programming[3] and stepwise refinement[4] made evident the advantages of top down development of software. Recent work has elaborated the top down development process, taking into account the work on software system structure discussed above, and clarifying the way that top down development proceeds. The most important contribution has been the recognition of the role played by *program specifications*. A program specification is a description of the behavior of a module, the behavior that will be depended on by any user, and that must be provided by any implementation. At any stage of design, the goal is to identify lower level abstractions useful in implementing the current level. As these abstractions are identified, their behavior is specified. The specification is given in advance of the implementation, and it provides a complete description of the interface of the module that will later implement the abstraction. The presence of the specification permits the question of how to implement the lower level modules to be deferred until a later stage of design.

As programming methodology has become better understood, there has been increasing interest in defining formal methods to support it. In particular, there has been much recent research in formal specification techniques,[5,6] which permit the specifications discussed above to be expressed in a formal language, i.e., one with a well-defined and unambiguous syntax and semantics. The advantages of such formal specifications are twofold: they are more precise and concise than informal specifications, and therefore may serve better the role of interface descriptions described above, and, in addition, given formal specifications, a formal proof that a module's implementation satisfies its specification is possible. However, formal specifications are more difficult to write than informal ones, and our current understanding of specification and verification techniques is insufficient to permit all useful abstractions to be described.

It is my belief that the present and near future construction of software systems can best be helped by popularizing the methodology. This can be done in a way that relies neither on a particular language, nor on the as yet incompletely understood specification and verification techniques. Instead, the following two ideas must be made clear to programmers:

1. What constitutes good modularity.
2. What constitutes good design practice.

Rules about good modularity are best explained by developing conventions, or better yet preprocessors, for existing languages in actual use; such conventions would permit a limited use of data abstractions and would prohibit current bad practices (such as non-local use of data). By expressing the rules in this way, they are explained in terms the programmers can understand, and furthermore, a tool is provided that helps in the development of a well-structured system. Good design practice then consists of top down decomposition into the kinds of modules that the programmers already understand, with emphasis on the role of (informal) specifications in the process, and especially on the necessity of specifications being given in advance of implementation. To aid programmers in understanding what specifications are, it is helpful to establish a specification standard which describes the kind of information that should be included in the specification, and gives a format for expressing that information. However, it is too early to require that specifications be given in a formal language. Formal methods in system design are not yet ready for practical use, but I believe use of the methods in an informal way can have considerable practical benefit.

REFERENCES

1. Liskov, B., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," *Comm. of the ACM 20, 8,* August 1977, pp. 564-576.
2. Wulf, W., R. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Trans. on Software Engineering SE-2,* 1976, pp. 253-264.
3. Dijkstra, E. W., "Notes on Structured Programming," *Structured Programming, A.P.I.C. Studies in Data Processing 8,* Academic Press, New York, 1972, pp. 1-81.
4. Wirth, N., "Program Development by Stepwise Refinement," *Comm. of the ACM 14, 4,* April 1971, pp. 221-227.
5. Guttag, J., E. Horowitz, and D. Musser, *Abstract Data Types and Software Validation,* Report ISI/RR-76-48, Information Sciences Institute, University of Southern California, Marina del Rey, August 1976.
6. Liskov, B. and V. Berzins, *An Appraisal of Program Specifications,* Computation Structures Group Memo 141-1, Laboratory for Computer Science, M.I.T., Cambridge, Mass., April 1977.

BEYOND FACTORIAL—Donald I. Good

Factorial is probably the single most thoroughly verified program ever. It, and other small examples such as stacks, typically are used in the literature to illustrate formal program verification methods, and this tends to create the impression that these small examples define the limits of effectiveness of formal verification. We must be careful not to paint an overly optimistic picture, but the view that formal verification methods are effective only on small examples, such as stacks and factorials, is equally overly pessimistic. Formal verification methods have been applied successfully to a variety of programs in the 1000-lines-of-code range and strong verification clauses are now being written into some software development contracts with the U.S. Government. Although there is much yet to be done, formal verification methods show considerable promise of becoming an effective approach to building reliable software in actual practice.

Basic research over the last ten years has done much to develop formal methods of program verification. The prob-

lem of actually verifying a 2000- to 5000-line program, however, is much larger than just verification methods, and we must begin to focus some of our efforts on solving the "whole" verification problem if verification methods are to become effective in actual practice. The breadth and depth of the problem that is faced by a person charged with producing a substantial verified program is substantial. It encompasses the following issues: verification methods, specification methods, programming methods, specification language, verification language, programming language, specification tools, verification tools, programming tools, development strategies.

Verification methods are the central issue, but are by no means the only issue. A verification demonstrates the consistency of a program and a specification. Therefore, attaining a verification requires using specification and programming methods that are both compatible with the verification methods. (By "programming methods" we mean the use of program constructs that are verifiable, as opposed to program "development strategies".) We must know what kinds of things can and should be specified about various program constructs and we must know how to verify that the program construct meets the specification. We also must know how primitive verifiable program constructs can be put together to form large verifiable programs. Given the basic methods, precise languages for expressing the specifications, program, and verification are required. The specification and programming languages express the actual program and its desired properties, and the verifications are done in the verification language which normally is some variation of the predicate calculus. The development of suitable languages involves all of the well-known problems of language design. Given these languages, effective tools are required to apply these languages to an actual problem. Without adequate tools, the languages and the methods they embody become effectively inaccessible. Finally, these tools must be used according to some reasonable strategy for step-wise program development. Just as independent compilation is of major importance in developing a large program, independent verification is crucial even for a program of modest size; thus step-by-step verification must follow a sound program development strategy. Attaining a verified program of substantial size requires effective solutions to all of these problems and, just as importantly, it requires that these solutions be effectively integrated so that they can be used together in a systematic and coherent development of a verified program. In actual practice, an inadequate solution to any of these problems or an ineffective integration of individually effective solutions can completely defeat the verification of a 2000- to 5000-line program.

Area Director:
Robert Balzer
Information Science Institute
Marina del Rey, California

Automatic programming

The term Automatic Programming is ill-defined. It means many different things to many people. It has covered everything from Fortran Compilers in the mid-50's to futuristic visions of machines which completely program themselves to accomplish a given goal, or are even capable of generating those goals themselves.

The three Automatic Programming sessions in this conference clearly cannot cover such a broad spectrum. Instead, they are focused on a set of small research projects each attempting to expand the computer's involvement in the programming process. Surely, such continued expansion of computer involvement in the programming process is the central thrust of all the efforts in this field.

The current state of this field can best be accessed by noting that programming is still largely a manual process. Significant progress has been made over the years, but it has largely been concentrated in the final stages of the programming process (such as language translators, test case generators, and debugging tools). What facilities exist which help us formulate and design systems, study tradeoffs between alternative designs, foresee the implications of design and implementation decisions, understand the effects of modifications, predict system performance, foresee system bottlenecks, or even prevent the introduction of bugs during implementation?

Such tools basically do not exist. Instead, programmers and systems analysts deal with these issues in their own internal ways. These internal methods are not accessible for analysis by others, are error-prone, incomplete, and difficult to transfer from one individual to another.

This deficiency has been recognized and responded to by the development of a set (actually several competing sets) of techniques for MANAGING the programming process. These techniques attempt to control the complexity of programming by advocating certain practices and styles while proscribing others, and by developing languages which encourage and/or enforce such direction. Certainly such endeavors have significantly improved our ability to control and manage these design and formulation processes occuring within our heads.

Such manual techniques can only go so far. The regularized (structured) environment that they have created must be augmented by various tools which

record the stepwise development of these processes as they occur within our heads, expose the decisions we are making, and enable us to see the implications of those decisions.

The challenge for the field of Automatic Programming is to extend the programming environment by constructing such tools which expose this stepwise development process and enable each stage to be analyzed, as the basis for the next step, without impairing the formulators' design flexibility. To accomplish this, there must exist a clear separation between a decision of what step to take next, which the formulator must control, and the carrying out of that decision, which the tool must perform to ensure that it is being accurately accomplished. In this way the integrity of the system being formulated is maintained as it moves from stage to stage. These tools thus ensure system integrity between formulation stages by automating the "clerical" operations which transform one stage into the next after a decision has been reached. Since a very large number of decisions must be made in any formulation process and some of them are very detailed, there may well exist some portion which the formulator relegates to the Automatic Programming tool. These Automatic Programming tools must also provide some analysis capabilities which enable the formulator to determine which decision to make next and how to make that decision.

All of the prototype tools covered in these three Automatic Programming sessions match this paradigm. They differ in the portion of the formulation process they address, the analysis capabilities available at each stage, and the degree of control the formulator maintains over the decisions which get made.

The first session concentrates on tools which address the process of specifying a program, while the second is composed of those which address the implementation of such a specified system. These two sessions are not intended to present new research results (these have been reported in various specialized conferences and workshops), but rather, to bring together and highlight representative work in this field for the NCC audience.

The final session, a panel, will attempt to access the extent and timeframe of the impact of such work within the computing community.

Informality in program specifications[1]

by ROBERT BALZER, NEIL GOLDMAN and DAVID WILE

Information Sciences Institute
Marina del Rey, California

A critical step in the development of a software system occurs when its goal-oriented requirements specification is transformed into a process-oriented form that specifies how the requirements are to be achieved. Only after this transformation has occurred can the feasibility of the system be analyzed and the consistency of the process specification with the requirements be verified. The key to this transformation is expressing the process-oriented specification abstractly so that its functionality is completely determined while the class of possible implementations remains broad.

We believe that such abstract process-oriented specifications are the key to rationalizing the software development process. Such specifications are, in reality, programs written in a very high level abstract programming language. As such, they could provide an effective interface between the two major software concerns: functionality and efficiency. These concerns should be decoupled so that the functionality of a system can be addressed before its efficiency has been considered. Once functionality has been accepted, it can be preserved while the system is optimized. Thus, since the abstract process-oriented specification is a program, its consistency with the requirements could be formally verified, informally argued, or tested by actually executing the specification. Furthermore, the end user could be given hands-on experience exercising the specification to see if it behaved as intended. Deviations and/or inconsistencies could be corrected in the specification before any implementation occurred.

Once the system's functionality has been accepted by the user, the efficiency of the system in meeting its performance requirements remains an issue. Such efficiency must be gained without altering the system's accepted functionality. We have argued elsewhere[2] that a computer-based tool can be built which guarantees maintenance of functionality while a program is optimized without sacrificing the programmer's ingenuity or initiative in determining how best to achieve efficiency.

In this work we are concerned primarily with the procedure by which such process-oriented specifications are obtained and with computer-based tools for their construction. We will begin by determining some attributes of a suitable process-oriented specification language, then examine why specifications would still be difficult to write in such a language. We will argue that the key to overcoming these difficulties is the careful introduction of informality based on partial, rather than complete, descriptions and the use of a computer-based tool which utilizes context extensively to complete these descriptions during the process of constructing a well formed specification. We will then present some results obtained by a prototype of such a computer-based tool on a few informal example specifications. Finally, we will discuss some of the techniques used by this prototype system.

REFERENCES

1. "Informality in Program Specifications," *IEEE Transactions on Software Engineering*, March 1978.
2. Balzer, Robert, Neil Goldman and David Wile, "On the Transformational Implementation Approach," *2nd International Conference on Software Engineering*, October 1976, p. 337.

The PSI program synthesis system, 1978—An abstract

by CORDELL GREEN
Stanford University
Stanford, California

This abstract will cover the current status of the PSI program synthesis system. PSI is a computer program that acquires high-level descriptions of programs and produces efficient implementations of these programs. PSI allows program specification dialogues using a mix of natural language, traces, examples, and a very high-level language. These specifications are used to construct a well-defined high-level program description of the desired program. This description is called a program model. The model is then refined into an efficient implementation of the program. PSI consists of several modules including an English parser-interpreter, trace and examples inference expert, domain expert, explainer, dialogue moderator, program model builder, coder and efficiency expert. The class of programs synthesized are simple symbolic computation programs in either LISP or SAIL.

The system design has been a group effort with the individuals responsible for each module as follows: parser-interpreter, Jerrold M. Ginsparg; trace and example inference and domain experts, Jorge V. Phillips; moderator, Louis I. Steinberg; explainer, Richard P. Gabriel; model builder, Brian P. McCune; coder, David R. Barstow and Juan Ludlow; and efficiency expert, Elaine Kant. The interested reader should see the short summary paper[1] for a list of references on these components.

PSI's operation may be conveniently factored into an acquisition phase and a synthesis phase. The acquisition phase is concerned with acquiring the program specifications and producing a program model. The synthesis phase then finds an efficient implementation in the selected target language.

An assumption used in the design of the acquisition phase of PSI is that there is no one best method for specifying all programs. So a mix of methods is allowed, ranging from a "conventional" very high-level language through English and traces. The English sentences are first parsed, then interpreted into fragments. The parser limits search by incorporating considerable knowledge of English usage. The interpreter is more specific to automatic programming, using program description knowledge as well as knowledge of the last question asked and the current topic to facilitate the interpretation into fragments.

Fragments form a loose description of program and data structure. The model builder must then apply knowledge of correct high level programs to convert the fragments into the model. The program model includes complete, consistent and executable (but slowly) high-level algorithm and information structures. The model builder processes fragments, checking for completeness and correctness, fills in detail, corrects minor inconsistencies, and adds cross-references. It also generalizes the program description, converting it into a form that allows the coder to look for good implementations.

Another input specification method is partial traces mixed with input and output examples. Partial traces of states of internal and I/O variables allow the inductive inference of control structures. The trace and example inference module infers loose descriptions of programs in the form of fragments, rather than programs themselves. This technique allows domain support to disambiguate possible inferences, and also separates the issue of efficient implementation from the inference of the user's intention. Application domain specific knowledge (e.g., knowledge about learning programs) is concentrated in the domain expert, which supplies other acquisition modules with domain support for disambiguation and program model completion.

The moderator guides the dialogue with the user by selecting or repressing questions generated by various modules. It attempts to keep PSI and the user in agreement on the current topic, provides a review-preview on a topic change, helps the user that gets lost, and allows initiative to shift between PSI and the user. The explainer module generates questions about and descriptions of program models as they are acquired, in order to help verify that the inferred program description is the one desired.

After the acquisition phase is complete, the synthesis phase begins. This phase may be viewed as a series of program refinements or as a heuristic search for an efficient program that satisfies the program model. The coder has a body of program synthesis rules that gradually transform the program model from abstract into more detailed constructs until it is in the target language. Both algorithm and data structures are refined interdependently. The coder deals primarily with the notions of sets and correspondences and can synthesize programs involving sequences, loops, simple input and output, linked lists, arrays, and hash tables.

The refinement tree effectively forms a planning space that proposes only legal, but possibly inefficient, programs. The efficiency expert reduces the search of the tree by several heuristics and by estimating the time-space cost product of each proposed refinement and then using the

branch and bound method. It also factors the program into relatively independent parts so that all combinations of implementations are not considered. An analysis for bottlenecks can allocate synthesis effort to more critical parts of the program.

In summary, we have formulated a framework for an automatic programming system and have a start on the kinds of programming knowledge that must be embedded therein. PSI is moderately successful in that a research prototype is currently running and has synthesized many different programs including a graph traversal program, simple storage and retrieval programs and learning programs.

REFERENCE

1. Green, Cordell, "A Summary of the PSI Program Synthesis System," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* Cambridge, Massachusetts, August 1977, pages 380-381.

Protosystem I—An automatic programming system prototype

by GREGORY R. RUTH
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Over the years people have come to understand certain functions of the programming process well enough to automate them—that is to replace those functions by programs. The most notable results were assemblers, compilers, and operating systems. Our knowledge and understanding of programming is once again reaching a level where a significant advance in automation is both necessary and possible. In particular, it is now possible (for certain simple application domains) to create a system that will take as input a specification for a user's application and will automatically design and code the desired data processing system. To demonstrate feasibility and gain insight into the issues and technology involved in creating such a system, a prototype automatic programming system (Protosystem I) for generating business data processing systems is currently being developed at MIT.

A MODEL OF THE PROGRAM WRITING PROCESS

The data processing system writing process may be conceived as a sequence of phases leading from the conception of a system to its implementation as executable machine code. A useful and plausible model for this sequence of phases is:

Phase 1: Problem Definition—The system specification is expressed in domain dependent terms in English that is understandable by the program developers.

Phase 2: General System Analysis and Design—The problem is reformulated in standard data processing terms and expressed as an instance of a known solvable problem class (in our case a subset of the class of all batch oriented dps's). Domain dependent policy and procedures are worked out in detail at this stage.

Phase 3: System Implementation—The system—the actual procedural steps and data representations and organizations—is constructed by intelligent selection from and adaptation of a number of standard implementation possibilities.

Phase 4: Code Generation—The design specifications are implemented in a high-level language (e.g., PL/I, COBOL) in a fairly straightforward, but not totally mechanical, way.

Phase 5: Compilation and Loading—A form is produced that can be "understood" and executed by the target computer.

These phases progress from a general notion of *what* is to be done by the desired system toward a detailed specification of *how* it can be accomplished. They also represent the classes of design and implementation problems involved in program writing, progressing from the most global and general considerations toward the most local and detailed issues.

Protosystem I seeks to automate the program writing process by automating and tying together the phases described in the model given above. That is, Protosystem I is designed in such a way that there are explicit parts or stages corresponding to each of the model phases. Each such stage embodies the knowledge and expertise of the human agent(s) for the corresponding phase, so that, given the same or similar input, it can intelligently produce comparable corresponding results.

The products of each stage should be sufficiently general and malleable so that further stages can have the maximum freedom in making their design contributions in the most effective and efficient ways. Consequently, we have chosen in Protosystem I to make the product of each stage a descriptive representation of the dps in terms of concepts and considerations appropriate for the next stage of development. In this way the programming process is conceived as the development of a succession of ever more precise system descriptions until, ultimately, a level is reached where every detail has been decided and the result is an executable computer program.

EFFICIENCY ENHANCEMENT IN SYSTEM DEVELOPMENT

To produce a credible and practical result an automatic programming system must perform a reasonable degree of

optimization. Current formal optimization methods pertain mainly to the compilation level; when the entire program development process is automated, new, additional types of optimization will have to be included.

The various possible types of optimizations fall quite naturally into categories that correspond to the program writing levels in our model. For instance, the combination of computations (for the sake of I/O efficiency) is something that should be considered during Stage 3 (system implementation) where the data and computational interrelationships among conceptual processing units are most evident. Problems involving efficient loop construction should be handled in a Stage 4.

THE DEVELOPMENT OF PROTOSYSTEM I

The Protosystem I effort began in 1971. Because of the difference in the natures of the technologies involved the work was divided into two parallel efforts: (1) a top-part-of-the-system effort (Phases 1 and 2), essentially of an Artificial Intelligence nature, concerned with natural language comprehension, model formation and problem solving, (2) a bottom-part-of-the-system-effort (Phases 3 and 4) addressing the problems of implementation and optimization of a program given an abstract relational specification (ultimately to be passed down from the top part of Protosystem I). The bottom part of Protosystem I has been completely implemented in the MACLISP language and is operational on the MIT LCS PDP-10. Research and development on the top part, being considerably more ambitious and novel, is not expected to cross the threshold of practical applicability for another five years, and so will not be discussed further in this article.

A structural diagram of the bottom part of Protosystem I is shown in Figure 1. The following sections give an explanation of its workings.

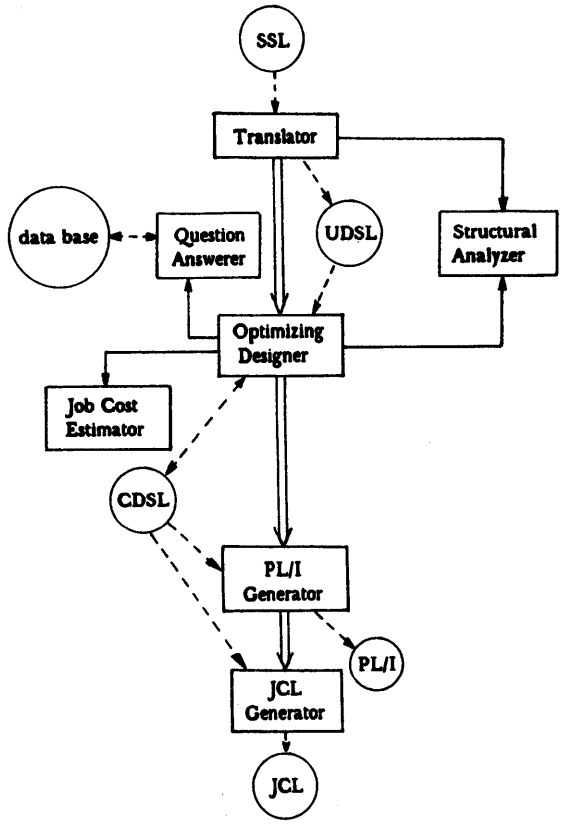


Figure 1—Protosystem I—Structure of the bottom part

THE PROTOSYSTEM I DATA PROCESSING SYSTEM MODEL AND THE SYSTEM SPECIFICATION LANGUAGE

Protosystem I handles a restricted but significant subset of all data processing applications: I/O intensive batch oriented systems. Such systems involve a sequence of runs or job steps that are to be performed at specified times. They are assumed to involve significant I/O activity due to repetitive processing of keyed records from large files of data. Systems such as inventory control, payroll, and employee or student record keeping systems are of this type.

A simple example of such a dps is a software system to perform the inventory and warehousing activities in the following case:

> The A&T Supermarket chain consists of 500 stores served by a centrally located warehouse. There are 4000 items, supplied by the warehouse, that these stores can carry.
>
> Every day the warehouse receives shipments from suppliers and updates its inventory level records accordingly.
>
> It also receives orders from the stores for various quantities of items. If for a particular item there is sufficient stock to fill all of the orders for that item, the warehouse simply fills the orders as made; but if there is insufficient stock it ships partial orders proportional to fraction of the total quantity ordered that is on hand.
>
> Inventory records are adjusted to reflect the decrease in levels.
>
> Finally, a daily check is made on the inventory levels of all items. If the level of an item is lower than 100 the warehouse orders 1000 more units of that item from the appropriate supplier.

In order for the bottom part of Protosystem I to implement such a data processing system application the basic aggregate data entities and their interrelationships must be determined. Consider the inventory updating activity of the second paragraph. There are three aggregate data entities involved: (1) the set of quantities received from suppliers, (2) the set of closing inventory levels for the previous day, and (3) the set of the updated inventory levels to be used for filling store orders. Such collections of similar data that are processed in a similar way are termed *data sets*. In

Protosystem I a data set is assumed to consist of fixed format *records* (e.g., one for the level of each inventory item). Associated with each record is a *data item* (e.g., the level of an inventory item) and *keys*. The key values of a record uniquely distinguish it (e.g., the inventory data set can be keyed by item since there is only one level [record] per item) and so can be used to select it.* If we call these data sets SHIPMENTS-RECEIVED, FINAL-INVENTORY and BEGINNING-INVENTORY, the relationship between BEGINNING-INVENTORY and the others may be described as:

For every item:

the beginning inventory level of that item
(i.e. the value of the data item for the record in BEGINNING-INVENTORY for that item)
is the closing inventory level of that item from the previous day
(i.e. the value of the data item in the record of FINAL-INVENTORY for the same item)
plus the quantity of that item received
(i.e. the value of the data item in the record of SHIPMENTS-RECEIVED for the item in question),
if any.

This relationship is expressed more succinctly in SSL (the System Specification Language):**

BEGINNING-INVENTORY IS FINAL-INVENTORY(1 DAY AGO) + SHIPMENTS-RECEIVED

The repetitive application of an operation to the members of a data set or sets such as this is termed a *computation*. The order of applications of the operation to the records of its input data sets by a computation is assumed to be unimportant to the user; in fact, he may think of them as being performed in parallel. However, every computation does, in fact, process its input serially, according to a particular ordering (chosen by Protosystem I) on their keys. Computations typically *match* records from different data sets by their keys (as above) and operate on the matching records to produce a corresponding output record. A computation may also *group* the members of a data set by common keys and operate on each group to produce a single corresponding output. Returning to our example, note that item orders can come from different sources (stores), so that both the item and the source of an order are needed (as keys) to distinguish it. To form the total of all orders for each item, a computation must group the orders by item and sum over the order amounts in each group. In SSL this would be expressed as:

TOTAL-ORDERS FOR EACH ITEM IS THE SUM OF THE QUANTITY-ORDERED-BY-STORE

Figure 2 shows the structure of the A&T inventory and warehousing data processing system in terms of computations (boxed) and data sets (unboxed). The complete SSL description of A&T dps is given in Figure 3. Note that in addition to the relational statements a list of data sets must be included to indicate the keys by which they are accessed.

THE TRANSLATOR AND THE DATA SET LANGUAGE

For the dps's which Protosystem I proposes to treat the calculations themselves are easily dealt with; the structuring and manipulation of the masses of data involved forms the major part of the Stage 3 implementation activity. Consequently, the development process at Stage 3 is data set oriented. Therefore, to facilitate the design process the SSL dps description is first analyzed from this point of view and re-expressed in a more appropriate medium, DSL (the Data Set Language). This reformulation is performed by the Translator module.

The determination of dps characteristics that can aid in the development of the dps design is made with the aid of the Structural Analyzer and included in the Translator's output description. This output is called the UDSL (Unconstrained Data Set Language) description, because most design details remain unbound (undecided) in it. As such it forms the skeleton of the dps description ultimately to be produced by Stage 3.

One useful piece of information determined by the Structural Analyzer is the set of *driving data set* candidates for each computation. A driving data set is an input data set that is guaranteed to have a data item for every tuple of key values for which the computation can produce an output. The computation, then, instead of having to loop over all possible combinations of values for the keys of the inputs, can be driven by the driving data set in that it only has to consider those key value combinations for which the driving data set contains records.

Another type of information the Structural Analyzer determines is directly related to our desire to specify data set organizations and orders and computation accessing methods and orders in such a way as to minimize the cost of operating the dps. Because a dps typically involves the repetitive application of *simple* calculations to large quantities of data we make the first-order approximation that the cost of operation is due entirely to data accessing (reading and writing). Our design, therefore, focuses on minimizing the total number of I/O events.

Accordingly, the Structural Analyzer also determines predicates that are the conditions under which a data item will be generated and under which a data item will be used by a computation. For example, a store will be shipped an item if (it is true that) that store ordered that item and there was sufficient inventory to fill the order; the order allocation step will use the inventory level for a particular item if some store ordered it. These predicates, together with basic information concerning the sizes of data sets in the dps, are used by the Question Answerer to determine the average and maximum sizes of files (proposed by the Optimizing Designer) and the average number of a file's records a computation will access.

* Thus, a data set is essentially the same as a Codd relation and its keys are what Codd calls primary keys.

** Implicit in this statement is that the addition operation is performed for each item and that if one of the operands is missing (e.g., if no chicken noodle soup was received today) it is treated as having a zero value.

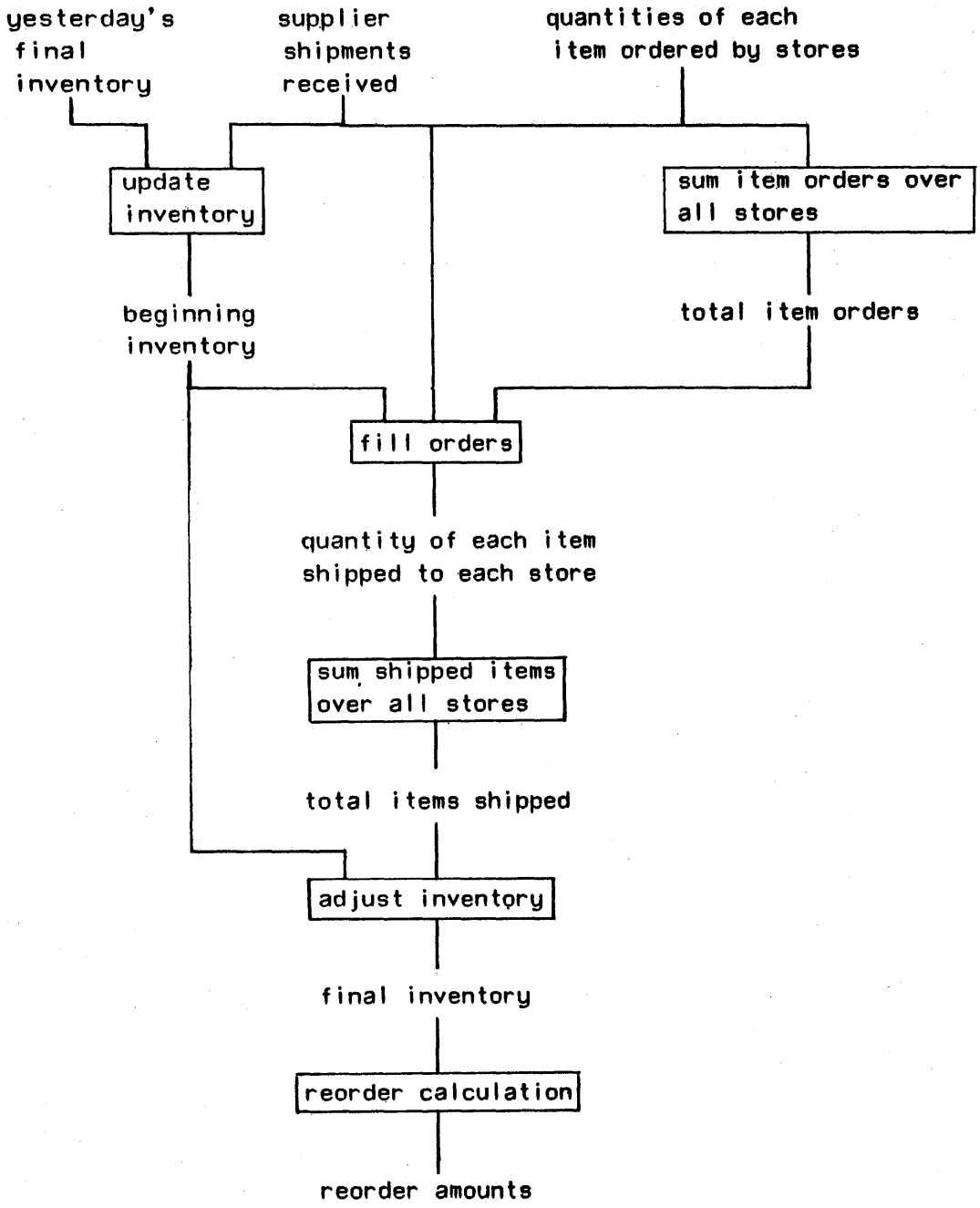


Figure 2—A&T inventory and warehousing system

THE DESIGN CRITERION AND THE JOB COST ESTIMATOR

The design criterion for Protosystem I is the minimization of the dollars and cents cost of running the final dps program on the target machine/operating system configuration. Because the dps's are assumed to be I/O intensive, as a first approximation, this can be equated with access minimization. An *access* in this sense is defined as the reading or writing of a single secondary storage block, which corresponds to a single operating system I/O event. In Protosystem I, for a particular data set a *block* consists of a fixed number of records.

With this approximation the relative costs of alternative dps design configurations can often be assessed without knowledge of the particular target configuration. But sometimes actual cost estimates, provided by the Job Cost Estimator, are necessary. This module must thus contain knowledge of the charging scheme and operating characteristics of the target configuration (in our case the OS/360 configuration). Optimization with respect to a different configura-

```
DATA DIVISION

FILE SHIPMENTS-RECEIVED
     KEY IS ITEM
     GENERATED EVERY DAY

FILE BEGINNING-INVENTORY
     KEY IS ITEM
     GENERATED EVERY DAY

FILE TOTAL-ITEM-ORDERS
     KEY IS ITEM
     GENERATED EVERY DAY

FILE QUANTITY-SHIPPED-TO-STORE
     KEYS ARE ITEM, STORE
     GENERATED EVERY DAY

FILE QUANTITY-ORDERED-BY-STORE
     KEYS ARE ITEM, STORE
     GENERATED EVERY DAY

FILE TOTAL-SHIPPED
     KEY IS ITEM
     GENERATED EVERY DAY

FILE FINAL-INVENTORY
     KEY IS ITEM
     GENERATED EVERY DAY

FILE REORDER-AMOUNT
     KEY IS ITEM
     GENERATED EVERY DAY


COMPUTATION DIVISION

BEGINNING-INVENTORY IS FINAL-INVENTORY(1 DAY AGO) + SHIPMENTS-RECEIVED

TOTAL-ITEM-ORDERS IS SUM OF QUANTITY-ORDERED-BY-STORE FOR EACH ITEM

QUANTITY-SHIPPED-TO-STORE IS

     QUANTITY-ORDERED-BY-STORE        IF BEGINNING-INVENTORY IS GREATER
                                      THAN TOTAL-ITEM-ORDERS

     QUANTITY-ORDERED-BY-STORE
     * (BEGINNING-INVENTORY / TOTAL-ITEM-ORDERS)
                                      IF BEGINNING INVENTORY IS NOT
                                      GREATER THAN TOTAL-ITEM-ORDERS


TOTAL-SHIPPED IS SUM OF QUANTITY-SHIPPED-TO-STORE FOR EACH ITEM

FINAL-INVENTORY IS BEGINNING-INVENTORY - TOTAL-SHIPPED

REORDER-AMOUNT IS 1000 IF FINAL-INVENTORY IS LESS THAN 100
```

Figure 3—SSL relational description for the A&T data processing system

tion and/or charging scheme would require the substitution of a new appropriately tailored module.

THE QUESTION ANSWERER

The function of the Question Answerer is to supply answers to questions from the Optimizing Designer about the average sizes (in records) of abstract aggregate data entities. Two examples of such data aggregates are a file and the collection of records in a file that are accessed by a particular computation. Each "question" sent to the Question Answerer is in the form of a predicate describing the conditions under which a record will be in the data aggregate in question.

The Question Answerer maintains a data base of all of the event probability and size information given by the user. When asked a question it attempts to find the associated size or probability directly. Failing this, it will try to calculate the probability of the event in question happening from those of its sub-events and its knowledge of event independence and correlation within the dps. If the information on hand is insufficient to answer the question, the Question Answerer obtains enough additional information from the user (through a flexible line of questioning) to do so.

THE OPTIMIZING DESIGNER

The Optimizing Designer is the heart of Stage 3; all of the other modules in this stage exist merely to serve it. When the translation from SSL to UDSL has been completed, control passes to the Optimizing Designer. This module is responsible for constructing job steps to implement computations and files to implement data sets. In particular its job is to:

1. design each keyed file—in particular its
   a. contents (information contained)
   b. OS/360 organization (consecutive, index sequential, or regional(2))
   c. storage device
   d. associated sort ordering (by key values)
   e. blocking factor (number of records per block)
2. design each job step of the dps—namely
   a. which computations it includes
   b. its accessing method (sequential, random, core table)
   c. its driving data set(s)
   d. the order (by key values) in which it processes the records of its input data sets
3. determine whether sorts are necessary and where they should be performed
4. determine the sequence of the job steps

The Optimizing Designer performs dynamic analysis (analysis of the operating behavior) on the dps to propose and evaluate alternative design configurations. Occasionally, static analysis (analysis of system structure and interrelationships) of such tentative configurations is also necessary, and this is obtained through calls to the Structural Analyzer. When additional information is needed to make evaluations and decisions the Question Answerer and the Job Cost Estimator are called.

All design decisions are made in an effort to minimize the total number of accesses that must be performed in the execution of the dps. There are three major techniques that the Optimizing Designer uses toward this end:

1. *Designing files and job steps to take advantage of blocking*—Accesses can be reduced if files are given blocking factors greater than one and if processing and file organizations are designed in such a way that the records of each block can be used consecutively.
2. *Aggregating data sets*—If two or more data sets that are accessed by the same computation are combined into one file (whose records have multiple data items) and processing is arranged so that a single record of the aggregate can be accessed where more than one record from each of the otherwise unaggregated files would have been accessed, accesses can be saved.
3. *Aggregating computations*—When two or more computations access the same data set and the orders in which they process the records of that data set are the same, it may be advantageous to combine them into a single job step. Then each record of the shared data set can be accessed once for all, rather than once for each computation.

These access minimizations techniques require that the key order of processing agree in a special way with the organization of the data being processed. Because different computations accessing the same data set may have different preferences for its organization, optimization of the type performed is necessarily a problem in global compromise.

The straightforward solution of evaluating the cost of every possible combination of assignments of sort order, device, organization, and access method for data sets and computations in every possible aggregation configuration to determine the least expensive is ruled out by the sheer combinatorics involved. Even with mathematical and special purpose tricks it would be impossibly slow.

To make optimization tractable a heuristic approach must be taken. First different kinds of decisions (e.g., choice of driving data sets, which objects to aggregate) must be decoupled wherever possible. Further decoupling must be judiciously introduced where it is not strictly possible, for the sake of additional simplicity. Such forced decoupling does not mean, though, that decisions that are in fact coupled are treated as if they were independent. The decoupled decisions are still made with a certain awareness of their effects on other decisions. Finally, as a first order approximation, the optimizer does what is reasonable locally, and then adjusts somewhat for global realities. While we make no claim that this approach will lead to the true optimum, it does produce good and usually near-optimal solutions for real and honest problems.

STAGE 4: CODE GENERATION

Stage 4 of Protosystem I consists of the PL/I and JCL Generator modules. The PL/I Generator takes the fully specified output of Stage 3 (the CDSL or Constrained Data Set Language description) as input and produces PL/I code for each job step. This involves the determination and arrangement of PL/I I/O specifics, the construction of the data processing loops, and the programming of the necessary calculations. The JCL Generator then writes IBM OS/360 JCL and ASP instructions for the I/O, administration and

scheduling of the compilation and execution of the dps job and job steps.

CONCLUSION

A model of the data processing system implementation process has been presented and a blue-print, based on that model, for automating the entire process has been developed. Protosystem I is a project to exhibit the feasibility of these ideas. Already, two of the four heretofore manual phases of the software writing process have been automated and are capable of producing acceptable implementations. The automation of the remaining two phases should easily fall within the realm of presently developing technologies within the next decade.

ACKNOWLEDGMENTS

The author wishes to thank Bill Martin, the originator of many of the ideas in this paper, and Mike Hammer, whose numerous comments, criticisms and suggestions were indispensable.

BIBLIOGRAPHY

1. Balzer, R., "Automatic Programming," Institute Technical Memo, University of Southern California—Information Sciences Institute, 1973.
2. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, 13,6, June 1970, pp. 377-387.
3. Early, J., "Relational Level Data Structures For Programming Languages," Computer Science Department, University of California, Berkeley, 1973.
4. Hammer, M., W. Howe, and I. Wladawsky, "An Overview of a Business Definition System," *ACM SIGPLAN Notices*, 9, 4, April 1974.
5. Hawkinson, L., "The Representation of Concepts In OWL," Fourth International Joint Conference on Artificial Intelligence, Sept. 1975.
6. Nunamaker, J. F. Jr., W. C. Nylin, Jr., and B. Konsynski, Jr., "Processing Systems Optimization through Automatic Design and Reorganization of Program Modules," *Information Systems* (Tou ed.), pp. 311-336, Plenum, 1974.
7. Ruth, G., "Automatic Design of Data Processing Systems," Third ACM Symposium on Principles of Programming Languages, Jan. 1976.
8. Ruth, G., "The New Question Answerer," Automatic Programming Group Internal Memo 21, MIT Laboratory for Computer Science, 1975.
9. Sussman, G., "A Computational Model of Skill Acquisition," MIT AI TR-297, MIT Artificial Intelligence Laboratory, August 1973.

DEDALUS—The DEDuctive ALgorithm Ur-Synthesizer*

by ZOHAR MANNA
Stanford University
Stanford, California

and

RICHARD WALDINGER
SRI International
Menlo Park, California

INTRODUCTION

Program synthesis is the automatic construction of programs to meet given specifications. These specifications constitute a high-level description of the desired program which expresses the purpose of the program, without indicating the method by which that purpose is to be achieved.

The specifications are expressed in terms of many constructs which are endemic to the particular subject domain of the desired program (e.g., numbers, sets, lists). Because these constructs are only intended to describe the purpose of the program and need not be computed, they can be of a much higher level than the constructs of any programming language (e.g., they can include logical quantifiers, set constructors, and other noncomputable operations). The specification language can correspond closely with the concepts a programmer actually uses in thinking about the problem.

The techniques we are developing are independent of the choice of a target programming language. The particular language we use in our examples and in our experimental system is a simple LISP-like language containing only basic numerical and list-processing operations, conditional expressions, and recursion. In considering the formation of programs with side effects, we extend the language to include assignments to variables, array elements, and other data-structure components.

Our basic approach is to transform the specifications repeatedly according to certain rules; each rule replaces one segment of a program description by another, equivalent, segment. The process continues until a description is obtained that is entirely in terms of the primitive constructs of the target language; this description is the desired program. The entire sequence of descriptions leading from the specifications to the final program is called a *program derivation.*

The transformation rules are guided by certain strategic controls, which ensure that they are applied only at the appropriate time. Many of the transformation rules represent knowledge about the program's subject domain; some explicate the meaning of the constructs of the specification and target languages; a few rules correspond to basic programming principles, which are independent of the particular subject domain or programming language.

The programs constructed by these techniques are guaranteed to be correct and to terminate, and require no separate verification phase. Up to now, we have not been concerned with the efficiency of the target program.

The techniques we develop are tested in the implementation of an experimental system called DEDALUS (DEDuctive ALgorithm Ur-Synthesizer). This system is implemented in the QLISP programming language, an extension of INTERLISP that includes pattern-matching and backtracking facilities.

The identification of the basic programming principles, their codification as transformation rules, and their implementation in our experimental system have constituted a major component of our effort. Some of the principles we have identified so far are:

- *Conditional formation*—This principle causes a case analysis to be introduced into the derivation, yielding a conditional expression (or test) in the ultimate program.
- *Recursion formation*—This principle introduces a recursive call into the ultimate program by observing when a subgoal to be achieved is actually an instance of the desired top-level goal.
- *Well-founded ordering*—The termination of the recursive programs formed by the above technique is ensured by constructing a well-founded ordering with the property that the arguments of the program's recursive calls are all strictly less than the program's inputs.

* This research was supported in part by the National Science Foundation under Grants DCR72-03737 A01 and MCS76-83655, by the Office of Naval Research under Contracts N00014-76-C-0687 and N00014-75-C-0816, by the Advanced Research Projects Agency of the Department of Defense under Contract MDA903-76-C-0206, and a grant from the United States-Israel Binational Science Foundation.

- *Procedure formation*—A subsidiary procedure is formed when a subgoal is found to be an instance, not of the top-level goal, but of a previously generated subgoal.
- *Generalization*—A generalized procedure is formed when two subgoals are found to be an instance of a third expression, which is somewhat more general than both.
- *Simultaneous goals*—In constructing a program to achieve two or more goals simultaneously, we first construct a program to achieve one goal, then modify that program to achieve the others as well, while protecting the condition that was already achieved.

Some of these principles are fairly well understood and have been incorporated into the DEDALUS system; others require further study. In the next sections we examine in some more detail our basic program-synthesis techniques, indicating which areas have already been implemented. Our treatment of these techniques will be extremely brief; further discussion of the same topics, at a more leisurely pace, appears in the authors' Stanford University-SRI International report, "Synthesis: Dreams⇒Programs" (November 1977).

GENERAL FRAMEWORK

Specifications

In designing the specification language, we have incorporated many constructs (e.g., the set constructor and the logical quantifiers) that facilitate the description of a program but that may not be included in the target programming language. We present below examples of specifications for simple programs using some of these high-level constructs.

A program *lessall(x l)*, to test if a number x is less than every element of a list l of numbers, is specified as follows:

lessall(x l)⇐**compute** $x<all(l)$
where x is a number and
l is a list of numbers.

In general, the specification construct *P(all(l))* denotes that the property P holds for every element of the list l.

The specification for a program to compute the greatest common divisor $gcd(x\ y)$ of two nonnegative integers x and y is

$gcd(x\ y)$⇐**compute** $max\{z : z|x\ and\ z|y\}$
where x and y are nonnegative integers and
$x\neq 0$ or $y\neq 0$.

The set constructor $\{u: P(u)\}$ denotes the set of all elements u satisfying the property P.

The *all* construct $(P(all(l))$ and the set constructor $\{u: P(u)\}$ are specification constructs that are nonprimitive (i.e., they are not in the target programming language). The synthesis task is to transform a description of the desired program, such as the specifications presented above, into an equivalent description that employs only primitive constructs of the target language.

The specification language is not fixed: as we consider new subject domains, we introduce new specification constructs accordingly.

Transformation rules

We use the notation

$t \Rightarrow t'$ if P

to denote that a subexpression of form t may be replaced by the corresponding expression t', provided that the condition P is true.

For example, the rule

$Q\ and\ true \Rightarrow Q$

denotes the basic logical principle that an expression of form "Q *and true*" may be replaced by Q. This rule has no conditions; it can always be applied.

The rule

$P(all(l)) \Rightarrow P(head(l))\ and\ P(all(tail(l)))$ if $not\ empty(l)$

expresses the fact that a property P holds for every element of a nonempty list l if it holds for the first element *head(l)* and for every element of the list *tail(l)* of the other elements. This rule imposes the condition that the list l be nonempty.

In the DEDALUS system, transformation rules are represented as programs in the QLISP language. The full expressive power of the programming language may be brought to bear in representing each rule.

Derivation trees

In developing a program whose specifications are

$f(x)$⇐**compute** $P(x)$
where $Q(x)$,

we establish the output description as a goal to be achieved, viz.,

Goal: **compute** $P(x)$.

Subgoals are derived from this goal by application of the relevant transformation rules. For example, in deriving the *gcd* program, we form the top-level goal

Goal 1: **compute** $max\{z: z|x\ and\ z|y\}$

By applying a transformation rule

$u|v\ and\ u|w \Rightarrow u|v\ and\ u|w-v$

we obtain the subgoal

Goal 2: **compute** $max\{z: z|x\ and\ z|y-x\}$.

If a transformation rule imposes a condition P, which must be true if the rule is to be applied, a subgoal of the form

Goal: **prove** P

must be achieved before the rule can be applied. For example, in developing the program *lessall*(x l) to test if a number x is less than every element of a list l of numbers, we have the top-level goal

compute $x<all(l)$,

which is obtained directly from the specifications; in attempting to apply the rule

$P(all(l)) \Rightarrow true$ if $empty(l)$

to this goal, we derive the subgoal

Goal: **prove** $empty(l)$.

From each subgoal that is derived, we can generate further subgoals by the application of more transformation rules. We thus construct a tree of goals and subgoals, which we will call a *program derivation tree*.

A subgoal **compute** S is already achieved if S consists entirely of primitive constructs of the target language. A subgoal **prove** P is achieved if P is the logical constant *true*. Such goals are terminal nodes of the derivation tree.

Let us now discuss some of the basic programming principles used in developing the derivation tree.

BASIC PROGRAMMING PRINCIPLES

Conditional formation

Many of the transformation rules impose conditions (e.g., l is nonempty, x is nonnegative) that must be satisfied for the rule to be applied. Suppose that in attempting to apply a particular rule, we fail to prove or disprove a condition P, where P is expressed entirely in terms of the primitive constructs of the programming language; in such a situation, the conditional-formation rule is invoked. This rule allows us to introduce a case analysis, and consider separately the case in which P is true and P is false. Suppose we succeed in constructing a program segment S_1 that solves our problem under the assumption that P is true, and another program segment S_2 that solves the problem under the assumption that P is false. Then the conditional-formation principle puts these two program segments together into a conditional expression

if P *then* S_1 *else* S_2,

which solves our problem regardless of whether P is true or false.

If we happen to generate the program segment S_2, say, without using the case assumption that P is false, then S_2 solves our problem regardless of whether or not P is true. In this case, no conditional expression is formed, and the program constructed is simply S_2. Thus, conditional expressions are generated only for truly relevant conditions.

The conditional-formation rule is among the best-understood of our basic programming principles and was among the first to be incorporated into the DEDALUS implementation. Other program-synthesis systems that form conditional expressions by case analysis have been implemented by Buchanan[1] and Luckham and Warren.[2]

Recursion formation

Suppose, in constructing a program whose specifications are

$f(x)$⇐**compute** $P(x)$
where $Q(x)$

we encounter a subgoal

compute $P(t)$

which is an instance of our *output specification*, **compute** $P(x)$. Because the program $f(x)$ is intended to compute $P(x)$ for any x satisfying its *input specification* $Q(x)$, the recursion-formation rule proposes achieving the subgoal by computing $P(t)$ with a recursive call $f(t)$. For this step to be valid, it must ensure that the *input condition* $Q(t)$ holds when the proposed recursive call is executed. To ensure that the new recursive call will not cause the program to loop indefinitely, the rule must also establish a *termination condition*, showing that the argument t is strictly less than the input x in some well-founded ordering. (A *well-founded ordering* is one in which no infinite strictly decreasing sequences can exist.) This condition precludes the possibility that an infinite sequence of recursive calls might occur during the execution of the program.

For example, the DEDALUS system derived the program *lessall* (x l), which tests whether a given number x is less than every element of a given list l of numbers. The specifications for this program are

lessall (x l)⇐**compute** $x<all(l)$
where x is a number and
l is a list of numbers.

In deriving this program, we develop a subgoal

compute $x<all(tail(l))$

in the case that l is nonempty. This subgoal is an instance of our output specification, with the input l replaced by *tail*(l); therefore, the recursion-formation principle proposes that we achieve the subgoal by introducing a recursive call *lessall* (x *tail* (l)). To ensure that this step is valid, the rule establishes an input condition, that

x is a number and
tail(l) is a list of numbers,

and a termination-condition that the argument pair (x *tail* (l)) is less than the input pair (x l) in a well-founded ordering. This termination condition holds because *tail* (l) is a proper sublist of l.

The final program we obtain is

lessall (x l)⇐*if empty* (l)
then true
else $x<head(l)$ *and lessall* (x *tail* (l)).

The recursion-formation principle is well-understood and is included in the DEDALUS implementation. The principle is the same as the "folding rule" of the Burstall and Darlington[3] program transformation system.

Procedure formation

Suppose in developing a program whose specifications are of the form

$f(x) \Leftarrow$ **compute** $P(x)$
where $Q(x)$

we encounter a subgoal

Goal B: **compute** $R(t)$,

which is an instance, not of the output specification **compute** $P(x)$, but of some previously generated subgoal

Goal A: **compute** $R(x)$.

The procedure-formation principle proposes that we introduce a new procedure $g(x)$ whose output specification is

$g(x) \Leftarrow$ **compute** $R(x)$.

In this way, we can achieve both Goals A and B by calls $g(x)$ and $g(t)$ to a single procedure. In the case that Goal B has been derived from Goal A, the call to $g(t)$ will be a recursive call; otherwise, both calls will be simple procedure calls.

For example, in constructing a program *cart* $(s\ t)$ to compute the Cartesian product s and t), of two sets, we are given the specification

cart $(s\ t) \Leftarrow$ **compute** $\{(x\ y): x \in s \text{ and } y \in t\}$
where s and t are finite sets.

In deriving the program, we obtain a subgoal

Goal A: **compute** $\{(x\ y): x = head(s) \text{ and } y \in t\}$

in the case that s is nonempty. Developing Goal A further, we derive the subgoal

Goal B: **compute** $\{(x\ y): x = head(s) \text{ and } y \in tail(t)\}$

in the case that t is nonempty. Goal B is an instance of Goal A; therefore, the procedure-formation rule proposes introducing a new procedure *carthead* $(s\ t)$ whose output specification is

carthead $(s\ t) \Leftarrow$ **compute** $\{(x\ y): x = head(s) \text{ and } y \in t\}$

so that we can achieve Goal A with a procedure call *carthead* $(s\ t)$ and Goal B with a (recursive) call *carthead* $(s\ tail(t))$.

The *carthead* procedure is constructed by the techniques we have already introduced. The final system of programs we obtain is

cart $(s\ t) \Leftarrow$ *if empty* (s)
then $\{\}$
else carthead $(s\ t) \cup$
cart $(tail(s)\ t)$,

where

carthead $(s\ t) \Leftarrow$ *if empty* (t)
then $\{\}$
else $\{(head(s)\ head(t))\} \cup$
carthead $(s\ tail(t))$.

The basic procedure-formation principle has been implemented, and the DEDALUS system can carry out this and other such derivations. However, our method for proving the termination of ordinary recursive calls does not always extend to the multiple-procedure case.

Generalization

Suppose in deriving a program we obtain two subgoals

Goal A: **compute** $R(a(x))$

and

Goal B: **compute** $R(b(x))$,

neither of which is an instance of the other but both of which are instances of the more general expression

compute $R(y)$.

Then the extended procedure-formation rule proposes that we introduce a new procedure, whose output specification is

$g(y) \Leftarrow$ **compute** $R(y)$,

so that we will be able to satisfy Goal A by a procedure call $g(a(x))$ and Goal B by a procedure call $g(b(x))$.

For example, in constructing a program *reverse* (l) to reverse a list l, we derive two subgoals

Goal A: **compute** *append*(*reverse*(*tail*(l))
cons(*head*(l)
nil))

and

Goal B: **compute** *append*(*reverse*(*tail*(*tail*(l)))
cons(*head*(*tail*(l))
cons(*head*(l) *nil*))).

Each of these goals is an instance of the more general expression

compute *append*(*reverse*(*tail*(l))
cons(*head*(l)
m));

therefore, the extended procedure-formation rule proposes introducing a new procedure *reversegen* $(l\ m)$, whose output specification is

reversegen $(l\ m) \Leftarrow$ **compute** *append*(*reverse*(*tail*(l))
cons(*head*(l)
m)).

This procedure reverses a nonempty list l and appends the result to m. Although the procedure solves a more general problem than the *reverse* program we actually require,

it turns out that the *reversegen* procedure is actually easier to construct. The final system of programs we obtain is

reverse (*l*)⇐*if empty* (*l*)
then nil
else reversegen (*l nil*)

where

reversegen (*l m*)⇐if *empty*(*tail* (*l*))
then cons(*head* (*l*) *m*)
else reversegen(*tail* (*l*)
cons(*head* (*l*) *m*)).

The generalization mechanism and the extended procedure-formation principle are just beginning to be formulated; the elaboration of these concepts is an important part of our projected effort. Generalization was proposed as a program-synthesis technique by Siklossy,[4] and is routinely performed by theorem-proving systems for proofs by induction (e.g., see Boyer and Moore[5]).

STRUCTURE-CHANGING PROGRAMS

In the discussion so far, we have been concerned with *structure-maintaining* (i.e., "side-effect-free") *programs,* which produce an output but effect no permanent change in the data objects of the programming environment. To produce *structure-changing programs,* which can effect such changes, we introduce structure-changing primitives, such as assignment statements, into our target language. To specify such programs, we admit new constructs into our specification language. Most notably, we employ **achieve** P to denote a program intended to make the condition P true.

All the principles we have applied to construct structure-maintaining programs may be adapted to construct structure-changing programs as well. However, certain new problems arise in the synthesis of structure-changing programs; among these is the *simultaneous-goal problem.*

In constructing a program to achieve two conditions P_1 and P_2, it is not sufficient to decompose the problem by constructing two independent programs to achieve P_1 and P_2 respectively. The program that achieves P_2 may in the process make P_1 false, and vice versa. Thus, the concatenation of the two programs will not achieve both conditions.

For example, suppose we want to construct a program to sort the values of three variables x, y, and z; in other words, we want to permute the values of the variables to achieve the two conditions $x \leq y$ and $y \leq z$ simultaneously. Assume that we are given the primitive instruction *sort*2(u v), which sorts the values of its input variables u and v. Then we can achieve each of our desired conditions independently by executing the program segment *sort*2(x y) and *sort*2(y z) respectively. However, the concatenation

*sort*2(x y)
*sort*2(y z)

of these two segments will not achieve both conditions simultaneously; in sorting y and z, the second segment *sort*2(y z) may make the first condition $x \leq y$ false.

To circumvent difficulties of this sort, we have introduced the following *simultaneous-goal principle*: To satisfy a goal of form

achieve P_1 *and* P_2,

first construct a program F to achieve P_1, then modify F to achieve P_2 while protecting P_1 at the end of F. The program-modification technique we employ is based on the "weakest-precondition operator" (Dijkstra[6]). A special "protection mechanism" (cf. Sussman[7]) ensures that no modification is permitted that destroys the truth of the protected condition P_1 at the end of the program.

To apply this principle to the goal

achieve $x \leq y$ *and* $y \leq z$

in the sorting problem, we first construct the program segment *sort*2(x y) that achieves the first condition. We then modify this program to achieve the second condition $y \leq z$. We cannot achieve this condition by inserting the instruction *sort*2(y z) at the end of the program, because (as we have seen) this modification violates the condition $x \leq y$, which we must protect.

However, our program-modification technique allows us to achieve a goal by inserting modifications at any point in the program, not merely at the end. In this case, the technique causes us to introduce the two instructions

if $y < x$ *then sort*2(x z)

and

if $x \leq y$ *then sort*2(y z)

at the beginning of the program segment. The modified program,

if $y < x$ *then sort*2(x z)
if $x \leq y$ *then sort*2(y z)
*sort*2(x y),

will achieve both conditions $x \leq y$ and $y \leq z$ simultaneously.

Similar techniques are employed by the system of Warren.[17]

The derivation of straight-line programs with simple side-effects is now fairly well understood; much work needs to be done on the derivation of structure-changing programs with conditional expressions and loops, and the derivation of programs that alter list structures and other complex data objects.

APPLICATIONS TO PROGRAMMING METHODOLOGY

Although the development of a practical program-synthesis system requires considerable research effort, certain applications of program-synthesis techniques to more restricted problems will be of more immediate practical value.

Various programming disciplines have been evolving that attempt to make the programming process simpler or more reliable. Let us consider several of these disciplines, to see where program-synthesis techniques may be applicable.

- *Structured Programming*—Like program synthesis, structured programming (cf. Dijkstra[8]) presents principles for deriving a program systematically from given specifications. However, the principles of structured programming are intended to guide a human programmer, whereas the principles of program synthesis are meant to direct a computer system. Nevertheless, we have found that some of the techniques we have developed for a program-synthesis system could well be employed by a human programmer. In particular, the recursion-formation principle is a better motivated guide for introducing a loop than the conventional structured-programming method for the same task.
- *Program Transformation*—In this approach (cf. Burstall and Darlington[3]), the programmer constructs a transparent program for his task, which is likely to be correct but which may be inefficient. This program is then transformed into an efficient equivalent program, which may be more difficult to understand. This transformation process is guaranteed to produce a program equivalent to the original. Program transformation may be regarded as a synthesis task in which the specifications are given in the form of a clear program in the target language. All the program synthesis techniques we have developed can be applied to program transformation as well.
- *Data Abstraction*—In this approach, the programmer expresses his program in terms of *abstract data types*, objects such as sets, queues, or graphs whose properties are well-defined but whose precise machine representation is left unspecified. When this program is complete, representations for its abstract data types are chosen and the program is transformed to replace the operations on the abstract data types by the corresponding concrete operations on the chosen representation (cf. Guttag et al.[9]).
- *Program Modification*—It is often observed that programmers spend more of their time extending programs that already perform some task correctly than they do in developing new programs. This process is particularly fraught with error, because in modifying a program, the programmer is likely to make some change that interferes with the program's original functioning. We have remarked that a program-modification technique was developed to support the simultaneous-goal principle. This technique can also be applied to perform independent program-modification tasks. The protection mechanism ensures that the modified program must still perform the task for which it was originally intended.

IMPLEMENTATION

It is difficult to develop or evaluate heuristic techniques without experimenting with an implementation. The DEDALUS system is a laboratory tool rather than a practical product. The system is implemented in QLISP (Wilber[10]), an extension of INTERLISP (Teitelman[11]) that includes pattern-matching and backtracking facilities. In this section, we will describe some of the special characteristics of our implementation without going into very much detail.

The specifications are expressed in a LISP-like notation. Thus, the output specification for the *lessall* program, which we wrote as

$$x<all(l),$$

is represented in the DEDALUS system as

```
(LESS X (ALL L)).
```

The output specification for the *gcd* program, which we wrote as

$$max\{z: z|x \text{ and } z|y\},$$

is represented as

```
(MAX (SETOF Z (AND (DIVIDES Z X)
                   (DIVIDES Z Y)))).
```

The target program is also expressed in LISP syntax.

The transformation rules are expressed as programs in the QLISP programming language. For example, the rule that we denoted by

$$Q \text{ and } true \Rightarrow Q$$

is represented by the QLISP program

```
(QLAMBDA (AND ←Q TRUE) $Q).
```

The rule

$$u|v \Rightarrow true \quad \text{if } u \text{ is an integer and } v=0$$

is expressed in QLISP as

```
(QLAMBDA (DIVIDES ←U ←V)
         (INSIST (PROVE ('(INTEGER $U))))
         (INSIST (PROVE ('(EQUAL $V 0))))
         TRUE).
```

Although the reader who is unfamiliar with the QLISP language may not understand all the details of the above programs, he may still observe that they are similar in form to the rules that they represent; the features of the QLISP language make this representation fairly direct. Because rules are represented as programs, we are allowed the full power of the programming language in expressing each rule.

The DEDALUS system currently contains more than a hundred such transformation rules. In expanding the system to handle a new subject domain, we simply introduce new rules.

The rules of the system are classified according to their *pattern*, their left-hand side. This pattern describes the class of subgoals to which the rule can be applied. Thus, the rules

$$u|v \Rightarrow true \quad \text{if } \ldots$$

and

$$u|v \Rightarrow u|v-u \quad \text{if } \ldots$$

both have pattern $u|v$, and can be applied to goals such as

compute $x|y+z$.

When a new goal is generated, the QLISP system retrieves those rules whose patterns match the form of the goal. This retrieval is facilitated by arranging the rules in a classification tree according to their patterns; thus the two rules above would be classified on the same branch of the tree. This mechanism allows us to avoid matching every rule in the system against each newly-generated goal.

If no rule matches the entire expression of a goal, its subexpressions are established as subgoals. If no rule matches any subexpression of a given goal, a *failure* occurs, and backtracking is invoked; the system attempts to find an alternate transformation that applies to a previous subgoal.

The QLISP pattern-matcher has special provisions for matching commutative functions. Thus, because the *and* operation is commutative, the rule

Q and true ⇒ *Q*,

represented as the QLISP program

(QLAMBDA (AND ←Q TRUE) $Q),

can be applied to goals of form "*true and Q*" as well as "*Q and true*". For this reason, commutativity rules such as

P *and* Q ⇒ Q *and* P

are not necessary in the DEDALUS system.

This kind of matching also occurs in the recursion-formation rule, in determining whether a new goal is an instance of some earlier goal. For example, in the actual synthesis of the *gcd* program, the top-level goal

compute $max\{z: z|x \text{ and } z|y\}$

is regarded as an instance of itself with the roles of x and y reversed, because the *and* function is commutative. The recursion-formation rule, therefore, is able to propose the recursive call $gcd(y\ x)$.

The final *gcd* program we obtain is

$gcd(x\ y) \Leftarrow$ *if* $y<x$
then $gcd(y\ x)$
else if $x \neq 0$
then $gcd(x\ y-x)$
else y.

Currently, the DEDALUS implementation incorporates the principles of conditional formation, recursion formation (including the termination proofs), and procedure formation, but not generalization or the formation of structure-changing programs. The techniques for deriving straight-line structure-changing programs were implemented in a separate system (see Waldinger[12]).

Representative samples of the programs constructed by the current DEDALUS system are the following.

Numerical Programs:

- the subtractive *gcd* algorithm
- the Euclidean *gcd* algorithm
- the binary *gcd* algorithm
- the remainder of dividing two integers

List Programs:

- finding the maximum element of a list
- testing if a list is sorted
- testing if a number is less than every element of a list of numbers (*lessall*))
- testing if every element of one list of numbers is less than every element of another

Set Programs:

- computing the union or intersection of two sets
- testing if an element belongs to a set
- testing if one set is a subset of another
- computing the Cartesian product of two sets (*cart*).

COMPARISONS WITH AUTOMATIC PROGRAMMING

It has been claimed (e.g., see Balzer[13]) that, for a complex programming task, it is unrealistic to expect the user to formulate complete, correct specifications for the desired program. In specifying an airline-reservation system, an operating system, or a spacecraft-guidance system, for example, we are unlikely to anticipate the desired behavior of the system in every possible situation. In some systems, the specifications for the program are formulated gradually through an extended dialogue between the user and the system. (See, e.g., Green[14] and Balzer et al.,[15] or the survey of Heidorn[16].) The dialogue is continued during the program-construction process, so that the user can aid in the design of the algorithm and resolve any ambiguities or inconsistencies the system might discover. Typically, these systems attempt to play the role of an expert programmer-consultant, and they tend to rely more on built-in or acquired knowledge of a particular subject domain than on deductive processes. By concentrating on basic programming principles, we have focused on general techniques that apply to any subject domain. The ultimate automatic programming system, of course, will combine specific subject knowledge with general deductive methods.

REFERENCES

1. Buchanan, J. R. and D. C. Luckham, *On Automating the Construction of Programs*, Technical Report, Artificial Intelligence Laboratory, Stanford University, Stanford, CA., May 1974.
2. Warren, D. H. D., "Generating Conditional Plans and Programs," *Proceedings of the Conference on Artificial Intelligence and Simulation of Behavior*, Edinburgh, Scotland, July 1976, pp. 344-354.
3. Burstall, R. M. and J. Darlington, "A Transformation System for Developing Recursive Programs," *JACM*, Vol. 24, No. 1, January 1977, pp. 44-67.
4. Siklossy, L., "The Synthesis of Programs from Their Properties, and the Insane Heuristic," *Proceedings of the Third Texas Conference on Computing Systems*, Austin, TX., 1974.
5. Boyer, R. S. and J S. Moore, Proving Theorems about LISP Functions, *JACM*, Vol. 22, No. 1, January 1975, pp. 129-144.
6. Dijkstra, E. W., "Guarded Commands, Nondeterminancy and Formal Derivation of Programs," *CACM*, Vol. 18, No. 8, August 1975, pp. 453-457.

7. Sussman, G. J., *A Computer Model of Skill Acquisition,* American Elsevier, New York, NY., 1975.
8. Dijkstra, E. W., A Discipline of Programming, Prentice-Hall, Englewood Cliffs, NJ., 1976.
9. Guttag, J. V., E. Horowitz and D. R. Musser, *Abstract Data Types and Software Validation,* Technical Report, Information Sciences Institute, Marina del Rey, CA., August 1976.
10. Wilber, B. M., *A QLISP Reference Manual,* Technical Report, SRI International, Menlo Park, CA., March 1976.
11. Teitelman, W., *INTERLISP Reference Manual,* Xerox Research Center, Palo Alto, CA., 1974.
12. Waldinger, R. J., "Achieving Several Goals Simultaneously," in *Machine Intelligence* 8: *Machine Representations of Knowledge,* (E. W. Elcock and D. Michie, eds.), Ellis Horwood Ltd., Chichester, England, 1977, pp. 94-136.
13. Balzer, R. M., *Automatic Programming,* Technical Report, Information Science Institute, University of Southern California, Marina del Rey, CA., September 1972.
14. Green, C. C., "The Design of the PSI Program Synthesis System," *Proceedings of the Second International Conference on Software Engineering,* San Francisco, CA, October 1976, pp. 4-18.
15. Balzer, R. M., N. Goldman, and D. Wile, "Informality in Program Specifications," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* Cambridge, MA, August 1977, pp. 389-397.
16. Heidorn, G. E., "Automatic Programming Through Natural Language Dialogue: A Survey," *IBM Journal of Research and Development,* Vol. 20, No. 4, July 1976, pp. 302-313.
17. Warren, D. H. D., *WARPLAN: A System for Generating Plans,* Technical Report, Department of Computational Logic, University of Edinburgh, Edinburgh, Scotland, June 1974.

Automatic representation selection for associative data structures

by PAUL ROVNER
University of Rochester
Rochester, New York

INTRODUCTION

Motivation

One of the persistent hopes for rapid progress in Computer Science is that standard solutions to software design problems will be developed, analyzed, and used. In particular, the hope for high-level languages has always been that enough low-level detail could be left to "the system" to allow a mere human to develop solutions for enormously complex problems. An ideal high-level language would allow a programmer to concentrate on the conceptual development of a solution for his problem, rather than on the design of a representation for his solution once he can express it.

Although existing high-level languages insulate a user from many details, they still require him to do low-level design for many problems, especially for complex ones. Such cases arise either because the language provides only low-level storage structures (records, arrays, pointers, lists) with which the programmer must encode his algorithm, or because it provides a general-purpose data structure for which an arbitrary (and usually sub-optimal) representation design choice has been made a priori (LISP,[1] SAIL[2]). Any single representation choice for a general data structure is very badly matched to some problems; the user of a general data structure must often spend much time designing a way to avoid unacceptable inefficiencies. Furthermore, good storage structure design requires much clerical work, for which a person whose goal is to solve a problem will usually have little time or enthusiasm. An automatic system, on the other hand, is well suited to the job of systematically comparing the cost and applicability of the many representations that are available for program data. In addition, an automatic system is less likely than a person to overlook possible representation candidates.

The problem

This paper explores one aspect of the representation selection problem: storage structure selection for associative data structures. By "associative data structure," we mean a collection of associations between abstract items. We express associations as ordered lists ("tuples") of items. Thus, an association between n given items is (conceptually) an n element list of them, called an "n-tuple". The role of each item in an association is identified by its position in the n-tuple.

Before going into specific detail about representation selection, we present a simple example which will be used to illustrate technical details as they arise. The example data base is a collection of relations among family members, their genders, and their places of residence. For the most part, we will be concerned with three-part associations (triples) such as:

```
Sexof·Abby=Female
Parentof·Eric=Roberta
```

or abstractly A·O=V. The A,O,V notation comes from considering the association as:

```
Attributeof·Object=Value
```

We view a collection of associations in the computer as a software "associative memory."[3] That is, one could retrieve any (3-part) association by specifying any one or two of its positions. Figure 1 describes these possibilities.

An associative language will contain operations like:

```
Make Parentof·Eric=Paul
```

and

```
Erase Homeof·ANY=Valhalla
```

which add or delete associations or sets of associations from the collection. There will also be operations to retrieve and make use of associative information like:

```
Print (Sexof·Peggy)
```

and

```
Foreach PERSON such that
  Homeof·PERSON=Rochester do
    Print (PERSON, "lives in Rochester").
```

Form	Example	Interpretation
A·O=V	Homeof·Paul=Valhalla	The association itself, if present
A·O=X	Parentof·Eric=PARENT	Parents of Eric
A·X=V	Parentof·CHILD=Paul	Children of Paul
X·O=V	RELATION·Eric=Rochester	Attributes linking Eric to Rochester
A·X=Z	Parentof·CHILD=PARENT	Child-Parent pairs present
X·Z=V	RELATION·PERSON=Male	All pairs of attributes and objects yielding Male
X·O=Z	RELATION·Abby=VALUE	All attributes of Abby and their values

Figure 1

In the latter case, PERSON is a variable that is repeatedly "bound" to the middle (Object) position of associations matching the pattern.

We are interested in how to implement a programming system that can do such operations efficiently. The difficulty, of course, is that each program is different and one cannot expect a single implementation to be best for all programs. What is required is an "intelligent compiler" which selects a different collection of storage structures for each program. The first such effort[4] was concerned primarily with sets and lists, which are simpler to deal with than associations. This paper explores the automatic selection problem for associations, and describes an experimental system (called the "selection system") which embodies our ideas. The discussion centers around associative data structures which fit into some "main" memory. (The problem for structures in several different kinds of memory is more complex.)

The following discussion illustrates the behavior of the system using the following simple program as an example:

```
BEGIN
   Attributes Parentof, Sexof, Homeof
   Item Constant Male, Female, Peggy, Abby, Rochester,...
   Item Variable CHILD, PARENT
     Make Parentof·Abby=Paul
     Make Parentof·Abby=Roberta
     Make Sexof·Abby=Female
     Make Homeof·Abby=Rochester
     Make Homeof·Thor=Valhalla
                .
                .
                .
(1)  Foreach CHILD, PARENT such that
        Sexof·CHILD=Male
         and Parentof·CHILD=PARENT do
         Print ("Sonof" PARENT "is" CHILD)
(2)  Foreach CHILD, PARENT such that
        Parentof·CHILD=PARENT
         and Homeof·PARENT=Rochester do
         Print (CHILD "writes home to Rochester for
         money")
END
```

Figure 2

First let us consider the even simpler program with the part in the box omitted. The program builds a collection of associations involving the three attributes. The "Foreach" part computes (parent,son) relationships in two steps. First it chooses an association of the form

Sexof·O=Male

and binds CHILD to the value of "O" in the association. It then uses that value to match an association of the form

Parentof·CHILD=V

and assigns the matching V to Parent, and prints. This is repeated until there are no more matching associations. We will see how the system chooses storage structures for this simple program.

A MODEL OF ASSOCIATIONS AND HOW THEY ARE USED

How should a program (in an associative language) be described for the purpose of designing a storage structure for its associative data structures? The first thing to note is that a program usually deals with more than one class of association. A class of associations is a group which will be represented the same way internally. In the example, there are two classes: one based on the Parentof relation and one on the Sexof relation. Since there are no searches on the Homeof relation (with the boxed statement omitted), the selection system would not include those associations at all. For each class of associations, the system will characterize the operations performed on that class and the possible values for unbound variables. In our simple case, each class has several MAKE operations and one SEARCH.

The differences among classes of associations are often reflected in the ways that the program accesses and modifies them. That is, a program often uses different parameter patterns for associative operations on different classes of associations. For example, the simple program includes a search for male children, i.e.,

⟨given: Sexof⟩·⟨variable to be bound⟩=⟨given: Male⟩

but not for the sex of a given child, i.e.,

⟨given: Sexof⟩·⟨given: Eric⟩=⟨variable to be bound⟩

For the "Parentof" association, on the other hand, there are two SEARCH parameter patterns (with the statement in the box included):

⟨given: Parentof⟩·⟨given: a child⟩=⟨variable⟩

and

⟨given⟩·⟨variable⟩=⟨variable⟩

In another case, a program may require access to attributes that relate two items, for example spatial relations between two objects in a 3-D scene: above, below, behind, or in-front-of.

Other reflections of the differences in associations are to be found in properties of the associations themselves. For example, the sex of a person has only one value, but a parent may have several children.

Often, the conceptual differences in the associations with which a program deals provide information that is useful for making low-level representation decisions. For example, a representation that immediately comes to mind for the Sexof attribute is a pre-defined (at compile-time) field of every PERSON record, to hold a code (one bit is enough) which specifies Male or Female.

If the program used only searches for the sex of a given child, such a representation would be just fine. But the program does not use the Sexof relation this way, but instead asks that people who are Male be enumerated. The above representation would require that all PERSON records (both male and female) be scanned for ones that are marked "Male". For this purpose, a list of male people might be a better representation. This example points out that classes of associations should be characterized both in terms of the operations of the program and the properties of the data.

Now that we have an intuitive understanding of what "class of associations" means, we will define it more precisely, and sketch how the automatic selection system determines the different classes of associations that a given program uses.

The process begins with a flow analysis phase, in which the system determines the possible run-time values of item variables at their points of use in the program, and the possible associations that could exist at run-time. The first approximation to the collection of classes of associations is one class for each MAKE statement, representing those associations that the MAKE statement would create (at run-time). For our simple example, each such class would contain one triple.

The next phase causes two classes to merge if any single SEARCH or ERASE operation could access associations from both classes. The basic idea is that each SEARCH or ERASE operation divides the store of triples into two subsets: those triples which could match the parameter pattern of the operation (subset A) and those triples which could not (subset B). If the A subsets for two operations intersect, the algorithm merges the subsets, and associates both operations with the result. This continues until there are no more mergers to perform. The result is a model of the (disjoint) sets of triples (called "classes of triples") that will exist when the program is run. Each class has an associated set of operations which access only the triples of the class. The use of this algorithm reflects an implicit assumption about computation cost: a representation that would require access to more than one storage structure to service a single associative retrieval operation is assumed a priori to be too expensive. For our simple example (with the statement in the box included), the algorithm would yield three classes of associations: one for each of the three attributes.

A LIBRARY OF REPRESENTATIONS

The automatic selection system knows how to deal with roughly twenty different storage structure representations, but these are all based on a few fundamental concepts. One representation of a set of triples is simply to store triples in a vector of entries three items wide and to search all entries for the answers to the questions in Figure 1. There are a variety of record techniques in which a fixed location in a block of storage is associated with an attribute, e.g., one could have records for people with a Sexof field, etc. Another common technique is to form a linked list of attribute-value pairs for a given object (a "property-list"). Other useful ideas include inverted lists (a link through all tuples which contain a given item), hash-coding,[5] and using one-bit fields to represent the presence or absence of a property.

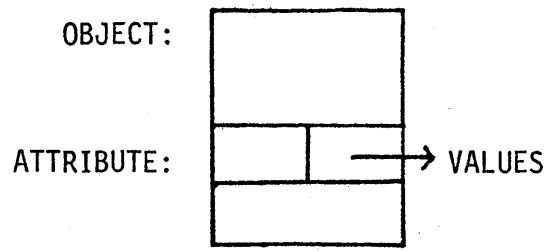
The selection system has a built-in library of representations for associations. The library is organized in two parts: "associative retrieval techniques" for servicing individual associative operations, and "associative representations" for common combinations of operations. The following discussion introduces the retrieval techniques and storage structures that provide access to triples, and describes the ways in which the storage structures can be varied and combined to synthesize "representations." This two-level organization is useful for proposing candidate representations: The analysis of which retrieval techniques are applicable to each associative operation on a class of triples need not be re-done for each (composite) representation that is considered. For example, once hash-coding is eliminated as a possible retrieval technique for an operation, no representations which would use hash-coding for the operation will be tried. Such methods for avoiding duplication of analysis effort are crucial for controlling combinatorial explosion in the selection process.

We have chosen four basic retrieval techniques upon which to base the example representations in the library: field selection records, property lists, inverted files, and hash tables. The diagrams in Figure 3 illustrate simple storage structures which implement these retrieval techniques.

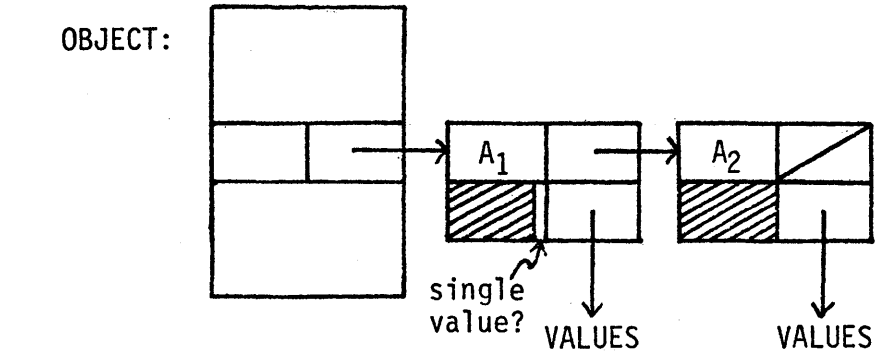
The library consists of twenty-three prototype representations which are based on these simple storage structures. We sketch these below: for a complete discussion, see Reference 6.

For the purpose of the discussion below, we will assume

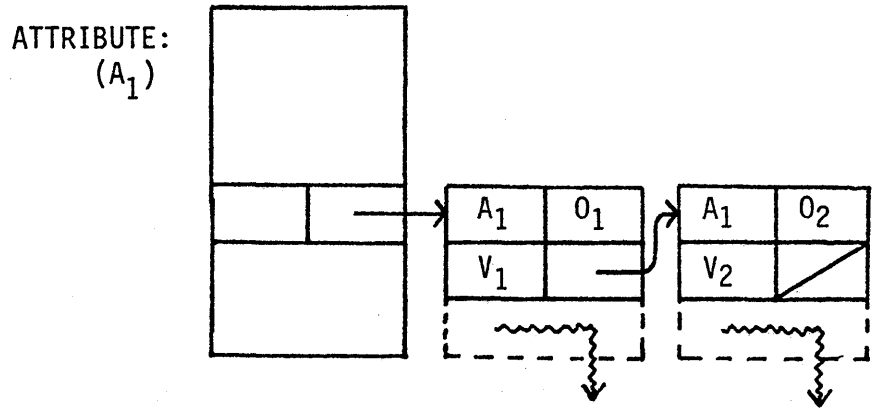
1. FIELD SELECTION RECORD



2. PROPERTY LIST SEARCH



3. INVERTED FILE SEARCH



4. HASH TABLE LOOK-UP

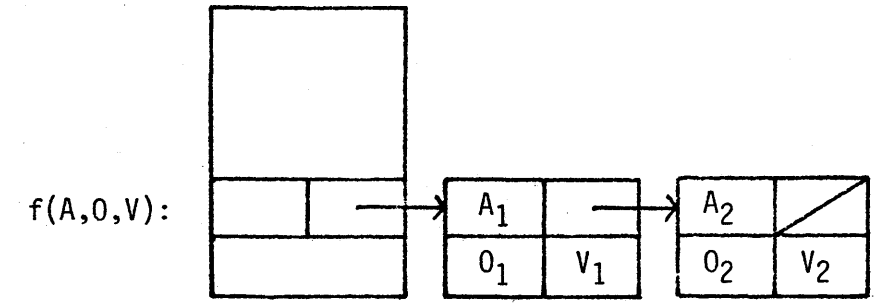


Figure 3

that each item has a unique (integer) "internal name." We will discuss using this integer as a hash-code operand, or as a key to various useful information about the item. Specifically, we will assume that the internal name of an item is a pointer to a block of contiguous storage cells, and refer to such blocks as "item descriptors." A typical item descriptor contains space for bit vectors and pointers to various lists that the selection system might allocate for triples that contain the item.

Field selection records

There are three variations, corresponding to the cases where the value field of the triple is binary (one bit), single-valued (e.g., the Homeof attribute), and multiple-valued.

Complex inverted list

This representation provides one sequential block of storage cells in memory for each triple (called "triple block"), with one field for each item. Each block can be threaded on either one, two, or three lists: one list for each item in the triple. The head of each such list is contained in an item descriptor.

P-List

A P-list is a list of (attribute-value) pairs. The head of each list is contained in the item descriptor for each object. There may be more than one value for each (attribute,object) pair.

Two argument hash-table

Two items (internal values of the attribute and object) are hash-coded (using a bucket hash) to locate a cell in a hash-table; the cell contains the attribute, the object, and a list of values. The cell can be threaded in either one or two lists: one list for each hash operand. The header for each such list resides in the item descriptor for the item which is the hash-operand. This variation is effectively a combination of a hash-coding retrieval technique and either one or two P-list retrieval techniques.

Another variation of the hash-table representation is to thread each value in an inverted list (of triples which have that value). Each of these representations combines a hash-coding retrieval technique with an inverted file retrieval technique.

A third variation is to use entries in the hash table to point to triple blocks, which can be threaded in a complex inverted list.

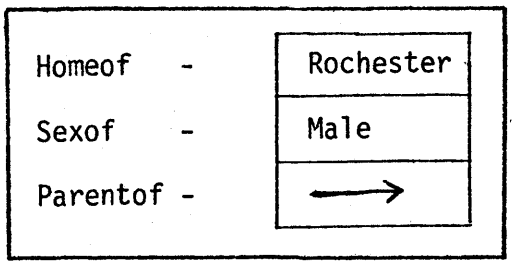
Three argument hash-table

The three item internal names are used as hash-coding operands. Each cell in the hash table represents one triple either by storing the three items, or by pointing to a triple block (which can be threaded as above). In addition, each cell can be threaded in one, two, or three inverted lists.

The above representations allow great flexibility in combining property-list, inverted list, and hash-coding retrieval techniques. This flexibility is most useful for dealing with ERASE operations, which must delete all paths to a triple in a complex representation when the triple is erased. In particular, certain of the above representations support back-pointers in their internal lists. The selection system makes use of this information. A more thorough discussion of the details is beyond the scope of this paper; the reader is referred to Reference 6.

Each retrieval technique in the library is associated with an "applicability" rule. This is an heuristic which is used to determine whether the access structure should be used to realize a given ERASE or SEARCH operation. Applicability rules usually reflect "common sense" (to a storage structure designer) rules, and are employed to limit combinatorial explosion in the selection process. These are useful constraints that represent pre-canned cost analysis decisions, like "blind search should be avoided whenever possible". Applicability rules are also used to detect relatively common special cases for which an unusual representation is particularly well suited.

For example, let us consider the basic record-style representation of a triple. A typical choice for the sample universe described above would be to have a record for each person with fields for his Sex, Home, and Parents. Since there may be two parents, the Parentof field might point to a set of parents, yielding something like the following structure for Eric:



There are additional complications, such as how to represent the set of parents. More to the point, this representation will not support an associative search such as:

Parentof·CHILD=Paul

without exhaustive search. The applicability rule for the record-style access structure would eliminate it as a candidate for the Sexof association because of the need for exhaustive search. Figure 4 contains a partial list of the computation cost tradeoff assumptions that underlie the heuristic applicability rules.

It is the nature of heuristic search that it is always possible to concoct situations in which a path that would turn out to yield an optimal solution is left untried. As for any heuristic method, the use of applicability rules is meant to achieve a

1. It is cost effective to maintain ordered lists of VALUE items in instances of the following access structures:
   Field Selection Records
   Property lists
   Two Operand Hash Tables
2. It is not worthwhile to consider an associative retrieval algorithm which requires that all items that could appear in a given operand position for a given class of triples be enumerated and tested.
3. The bookkeeping overhead for dealing with storage blocks for Field Selection Records whose lengths might vary at run-time is not cost effective.
4. It is cost effective to maintain an ordering relation on the elements of a property list access structure.
5. It is not worthwhile to consider an associative retrieval algorithm which requires the examination of all triples in a property list access structure for a given item.
6. It is cost effective to maintain an ordering relation on the elements of an inverted list access structure.
7. It is not worthwhile to consider an associative retrieval algorithm which requires the examination of all triples in an inverted list access structure for a given item.
8. It is cost effective to use the necessary storage space and execution time to maintain explicit "back pointers" in an inverted list access structure if it would otherwise be necessary to search the list from its beginning to find the list element which precedes a given one.
9. A hash coding scheme which uses chaining for collisions is as good as any other hash coding scheme.
10. It is not worthwhile to consider an associative retrieval algorithm which requires preliminary searching to locate the buckets in a hash table which might contain answers.

Figure 4

compromise between optimal solutions at tremendous expense and poor solutions that are easy to find.

STRATEGIES FOR PROPOSING REPRESENTATIONS FOR THE TRIPLES OF A GIVEN PROGRAM

Proposing non-redundant representations

This section describes how the selection system uses the model of a class of triples and the information in the library of representations to propose storage structures to represent the class. Proposal proceeds in two phases. First, each operation is analyzed to determine a set of applicable techniques. Next, an attempt is made to find representations which provide an applicable technique for each operation of the class. Heuristic rules are used to eliminate inefficient representations.

For each operation on the class, each technique in the library is examined. If its applicability criterion is satisfied for the operation, it is added to the set of techniques that are applicable to that operation. A technique may apply to an operation in more than one way. For example, suppose we had a SEARCH operation of the form:

⟨given⟩·⟨given⟩=⟨variable⟩

(like Parentof·⟨given child⟩=⟨variable to bind⟩)

and we were considering the inverted file technique. We could either keep a list of triples for each item which appears as ATTRIBUTE (e.g., Parentof), or we could keep a list of triples for each item which appears as OBJECT (e.g., child). In the first case, we would use the given ATTRIBUTE to locate the list, and then search it for triples which have the given OBJECT. In the second case, we would use the OBJECT to locate the list, and then search it for triples which have the given ATTRIBUTE. Locating such an inverted list of triples is a simple computation: the pointer to the list would be contained in a known field of the ATTRIBUTE's (OBJECT's) item descriptor. The internal name of an item is a pointer to its item descriptor; the selection system would assign one field of the descriptor to contain the pointer to the inverted list.

In the above example, the two options are represented by the selection system as a set of permutations of the triple operand pattern, as follows:

{(AOV),(OAV)}

The convention for denoting variations of the inverted file technique is that the first letter in a permutation indicates the position by which the list is accessed, and the second indicates the position by which the list is ordered. The two options here are "the list is accessed by A (i.e., ATTRIBUTE), and ordered by O (i.e., OBJECT)," and "the list is accessed by O, and ordered by A." For our simple example, these correspond to keeping a single list of all (child,parent) pairs, ordered by the internal name of the child, and keeping an unordered list for each child of his parents.

After each operation in a class is associated with all techniques that can be used to realize it, the second phase of proposal occurs: finding candidate storage structures that provide techniques for all operations of a class. In our example (with the statement in the box included), the Parentof class has two search operations, with the following forms:

⟨given⟩·⟨given⟩=⟨variable⟩

and

⟨given⟩·⟨variable⟩=⟨variable⟩

A slightly more complex case is a class with several operations where all operations have the same set of applicable techniques, and for each of these techniques the same set of permutations. This will occur, for example, if all operations of the class have the same form (i.e., the same pattern of ⟨given⟩,⟨variable⟩, or ANY parameter descriptions). In such a case we treat the set of operations as a single operation, and the task is the same as described above. All operations of the class will use the same associative retrieval technique.

In the general case, there are several forms of SEARCH operations which retrieve associations, and the sets of applicable techniques are not the same for all operations. Even in such a case, the intersection of these sets may not be empty. That is, there may be at least one technique which is applicable to more than one operation. The general prob-

lem is to find ways to decompose a set of operations into subsets each of which can be serviced by one associative retrieval technique.

Each representation in the library is cross referenced by the retrieval techniques that it provides. The algorithm for proposing representations for a decomposition is sketched below:

> Each representation which provides the required number of techniques is considered. Each applicable technique of each subset of operations in the decomposition is examined. If it is a technique provided by one of the representations, then the applicability rule for each such representation is evaluated. The evaluation yields a (perhaps empty) set of "candidate representations" to be proposed.

Each such "candidate representation" is a data structure which instantiates one of the representations in the library. That is, it specifies which retrieval techniques are to service which (subsets of) operations, and how (i.e., the operand permutation). In other words, an applicability rule for a representation specifies acceptable matches between the techniques that are provided by the representation and the techniques that are applicable to the given operations.

To avoid duplication of effort, the selection system treats similar candidate representations as a unit whenever possible. Specifically, candidate representations which differ only in their operand permutations are usually treated as a unit. For instance, the inverted file representation for the example above has two permutations which would be treated as a unit until final cost analysis is done.

The system cannot always treat representations which differ only in their operand permutations as a unit. Aside from a few such special cases, however, differences in operand permutations are ignored until cost analysis is done, thus saving both space and time in the early phases of automatic selection.

Proposing redundant representations

The discussion above describes a method for proposing candidate representations for a class of triples where each triple in the class is stored in memory only once. In such a representation, each triple will be represented by either one triple record (with perhaps several threads), or one VALUE cell (perhaps in a list of them), or one field of an item record.

The selection system also considers representations in which triples are stored more than once. The advantage of keeping multiple copies is that each copy can serve a special purpose for which its storage structure can be custom designed. In a case where there is no single representation which will service all the operations on the class efficiently, there may be a way to decompose the set of operations into subsets for which efficient special purpose representations can be found. It is more likely that an efficient storage structure can be found for a small set of associative operations than for a large set. Depending on the cost tradeoff between storage space and execution time, such a decomposition might be "cheaper" than a monolithic representation, even though more storage space is required. For example, a membership relation ("Memberof·Set=Element") between sets and their elements might be represented both as a list for each set of its elements and as a list for each element of the sets to which it belongs. Such a representation would provide speedy access both for queries about the elements of a given set:

Memberof·⟨given⟩=⟨variable to bind⟩

and for queries about the sets to which a given element belongs:

Memberof·⟨variable to bind⟩=⟨given⟩

To propose redundant representations, the selection system uses the method for proposing non-redundant representations as a subroutine. The problem is to discover decompositions for which efficient redundant representations can be found. The method is based on enumerating the ways of partitioning the set of ERASE and SEARCH operations on a class of triples. Roughly, we look for ways to decompose the set of operations on a class into a collection of subclasses, each of which can be analyzed for applicable storage structure representations (using the methods described above) as if it contained the only operations on the class. The idea is to capture in the notion of "subclass" a way to identify groups of operations for which efficient storage structures exist. The result of the analysis of a given class of triples is a collection of candidate redundant representations, each having one "efficient" (see below) storage structure for each subclass in one partition. Thus, for a particular partition (i.e., set of n subclasses), each proposed representation requires n copies of each triple: one copy in each of n storage structures. Each MAKE operation for the class of triples must create its triple in each of the n storage structures, and each ERASE operation must remove triples which match its operand pattern from all n storage structures (problems of critical races and deadlock do not arise).

In our example of a redundant representation for a set membership relation, each association would be stored twice: once as an entry on the list of elements in the given set, and once as an entry on the list of sets which contain the given element.

In any representation, each execution of a SEARCH operation must find all triples which satisfy the operand pattern of the SEARCH. We assume that each SEARCH operation is associated with exactly one subclass. That is, it will find all its answers from one storage structure. Thus, each subclass must service all MAKE operations for the class. For consistency, each storage structure will contain all the triples of the class (one could imagine clever ways to relax this requirement, but we do not consider them here).

Unless the rules for acceptable redundant representations are more restrictive than the rules for non-redundant representations, the number of candidate representations will be very large, and the cost of the overall analysis will be huge. There is always a way of using our library and

applicability rules to design a storage structure to service a given set of associative operations. If everything else fails, a complex inverted list representation could be used, with a separate thread for each operation. Thus, every possible subclass would be acceptable, and each partition of the class of triples would be plausible. If several representations were applicable to each subclass, things get worse. If we consider the fact that the number of applicable redundant representations for a partition is the product of the numbers of representations for its subclasses, we can begin to see that unless the applicability rules are tightened for subclasses, there could be very many candidate redundant representations indeed.

To limit the combinatorial explosion of candidate representations that use multiple copies of the store of triples, we have chosen a specific notion of efficiency and an heuristic rule to decide when to propose such representations. In general, we consider a storage structure to be inefficient for an operation if significant searching is required to service the operation. Many of the cost tradeoff assumptions that are used by the selection system are based on this idea, and they provide (crude) thresholds for early detection of representations that are likely to be eliminated eventually in the detailed cost analysis. The most expensive kind of searching is "unrestricted" searching; it is this sort that the applicability rules disallow. For example, we do not even consider a record technique for a SEARCH operation which has only the ATTRIBUTE given; a search through all possible items which could be the OBJECT of a triple would be required to perform the SEARCH. It is convenient to distinguish four types of search:

1. unrestricted
2. loosely restricted
3. tightly restricted
4. fully restricted

An example of "unrestricted" search is given above. By "loosely restricted" search, we mean (for example) search for elements which have a given attribute in a list of (attribute-value) pairs. By "tightly restricted" search, we mean (for example) search for a given item in a set of them (e.g., is Paul a Parentof Eric?). By "fully restricted" search, we mean access to answers via computation rather than search, e.g., selection of a given field of a given record, or hash-table look-up.

The cost tradeoff assumptions used in proposing non-redundant representations lead to applicability rules which eliminate representations that require unrestricted search. Loosely restricted search is allowed for non-redundant candidate representations, but not for redundant ones. In other words,

> We assume redundant representations which require loosely restricted search to be inefficient enough to be eliminated a priori as candidates for final selection. Intuitively, it is only worth the price of multiple representations if searching can be tightly (or fully) restricted.

As redundant representations are synthesized for a class of triples, they are tested for loosely restricted search requirements. If such search is required, the representation is not proposed as a candidate.

Now we consider the problem of proposing candidate redundant representations. The crude way to do it is to generate all possible ways of partitioning the set of SEARCH and ERASE operations of the class of triples. For each partition, each subclass would then be analyzed to determine applicable non-redundant representations. The cross product of the sets of representations for the subclasses of a partition would be the applicable redundant representations. The partition with one subclass (i.e., non-redundant) would fall out.

The inherent symmetry of the problem leads to combinatorial duplication of effort; early detection of situations that have already been analyzed is important.

The analysis deals with subclasses, which are sets of SEARCH, ERASE, and MAKE operations, and partitions, which are (initially disjoint) sets of subclasses. Each associative operation corresponds to an expression in the source program, and is represented by the corresponding expression node in the syntax tree of the program.

The selection system uses a central data structure to keep track of the subclasses and partitions which it analyzes. This data structure (called the "set dictionary") is similar in motivation to the "discrimination net" of QLISP:[7] it associates a unique descriptor with each set that is entered.

The set dictionary is used by the selection system to avoid duplicating:

1. The computation of applicable non-redundant representations for each unique subclass (i.e., set of operations).
2. The computation of redundant representations for each unique partition.

Whenever a new subclass or partition is analyzed, it is entered into a set dictionary and the result of the analysis is recorded with the entry. A sketch of the method for proposing candidate redundant representations is presented below:

> A recursive function in the selection system generates the partitions of a set of operations, using another recursive function to generate the subsets (subclasses) of the given set. Each subset that is generated is looked up in the set dictionary. If not found, it is assigned a (new) descriptor, entered, and tested to determine whether applicable representations exist (this test considers cost tradeoff assumptions for redundant representations. The new descriptor is associated with the result of the test. If it is found in the set dictionary, then it has already been tested. If the result of the test indicates no applicable representations exist, then the next subset is generated and the process continues. Thus, no subset is considered further unless it has applicable representations; i.e., once a subclass is analyzed and found to fail, no partition which contains the subclass will be considered. Furthermore,

because of the structure of the library, if a subclass fails, any subclass that contains it will fail. In other words, if there is no efficient representation for a given set of n operations, there will not be one for any larger set that contains the n operations. Thus, supersets of failed classes can be eliminated automatically.

A method similar to the one described above for subclasses is used for avoiding duplication of analysis effort for partitions. A partition is a set of subclasses. Each such set has a unique entry in the set dictionary. Thus, there are two kinds of entries in the set dictionary: sets of operations (subclasses) and sets of subclasses (partitions). After a partition is analyzed the first time, its descriptor is marked, and subsequent attempts to analyze it are short-circuited.

The cost of avoiding duplication of analysis effort is the set dictionary look-up time; the large relative cost of analyzing a subclass makes this overhead worthwhile. Another advantage of using the set dictionary is that storage space is conserved: only one copy of each unique set is kept, and only one copy of the analysis results is kept.

HOW TO SELECT A GOOD REPRESENTATION FROM THE CANDIDATES

In this section we show how to estimate the cost of each candidate representation and choose the cheapest one. We consider two components of computation cost: execution times (in micro-seconds) and storage space (in 36-bit words). For a given program, we are interested both in the execution time required to service its associative operations, and the storage space required for its associative data structures. We use three kinds of parameters to characterize each class of triples in a given program:

1. the average sizes of substructures of the class of triples (e.g., For a given A and O, how many V's will there be on average?).
2. probabilities of occurrence of searches that fail and changes that are redundant.
3. relative frequencies of statement executions.

The computations for estimating the space and time costs of a given candidate representation for a given class of triples are based on evaluating formulas which are expressed in terms of these parameters. Each representation in the library is supplied with two functions: one for estimating storage space cost, and one for estimating execution time cost. The arguments to a space function specify a set of associative operations and for each operation the associative retrieval technique that will be used to realize it if the given representation is chosen. This information is used by the space function to identify the parameters for estimating storage space cost.

A function for estimating time cost is a little more complicated. The arguments are the same, but the time function must sum the estimated contributions to total execution time of each of the operations. For each operation, this estimate is the product of two terms: the number of times the operation was executed in the sample execution and the estimated cost of one execution of the operation for the candidate representation. The estimated time cost of one execution of the operation is determined by a formula that is specific to the use of the indicated associative retrieval technique in the indicated representation. Each representation in the library is provided with formulas for each of its associative retrieval techniques. There is one such formula for each associative operation to which a technique applies. Thus, for each operation, the arguments to the time function are used to select a formula and to identify necessary parameters.

For our example, the space cost formula for a record representation for Parentof and the time cost formula for a search operation of the form

$$A \cdot O = X$$

are shown in Figure 5. The space formula represents the requirement that there be a field of each record (0.5 cells) for each set-valued attribute (e.g., Parentof), and that space for the set of values is needed. In our example, there is only one attribute (Parentof), hence NA=1. NO (number of objects) represents the number of people. NVALS(A,O) represents the average number of parents per person.

The flow chart for the search algorithm is also shown in Figure 5. The time cost formula represents the cost of the algorithm. C1, C2, and C3 are constant terms; to a good

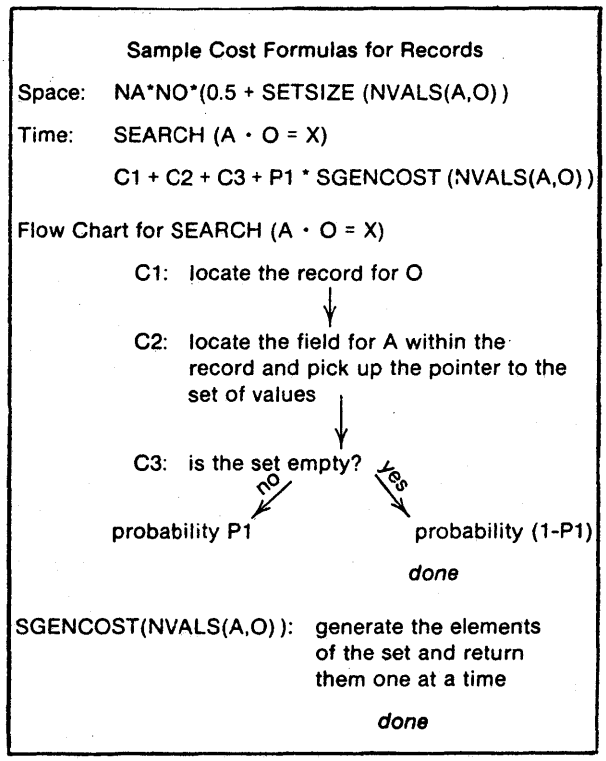


Figure 5

approximation they depend only on the machine and on basic design decisions and can be determined a priori. P1 represents the probability that the search succeeds. The call on the SGENCOST function represents the execution time required to generate the elements of a set of a specified size (NVALS(A,O)). The values of P1, NO, and NVALS(A,O) are determined from answers to questions that the system asks. The choice of which questions to ask is based on which cost formulas are being used in a given run of the selection system.

Parameters that depend only on the representations in the library (e.g., C1, C2, C3) are determined once when the library is defined, and built into the library. Their values are always available. Parameters that depend on the program are determined from information supplied by the programmer in response to questions that the selection system asks.

For example, the selection system asks three questions about the use of the Sexof relationship in our simple example. They are:

1. How many males are there likely to be? Sample answer: 5.
2. How often will the Sexof search fail to match? Sample answer: Never.
3. How often will a redundant MAKE occur? Sample answer: Never.

Not all parameters are needed: only ones required to evaluate formulas that are relevant to the representation candidates for a class of triples. Moreover, some parameters can be computed from others. For example, if the answers to

"For a given A, how many O's (AVG) will have at least one V?"

and

"For a given A and O, what is the average number of V's?"

are known, then the selection system can deduce the average number of triples, since it can determine (by asking a question, if necessary) the average number of items that can appear in a triple of the class in a given position. Thus, any two of the above three answers determine the third. The system makes use of such dependencies to derive needed parameters from available ones if it can. If it can't, it will ask a question.

We should say a word here about our assumptions regarding parameters. In any system that does automatic representation selection, a model of the expected behavior of the program is necessary. To make the analysis tractable, we made many intuitive decisions about the appropriate level of detail for cost formulas and their parameters. Two major assumptions about program behavior characterize our model and underlie our work on cost analysis:

1. It is in general impossible to tell for an arbitrary program precisely how it will behave when it is executed. Instead, we depend on the relative frequencies of statement executions in one example execution, and on estimates given by the programmer about program behavior and data structure size. We assume that the programmer has good enough intuitions about his program to provide a good general characterization. After all, he would have to use such intuitions if he were to design a representation without the help of an automatic system.
2. To really do an optimal selection, we would need to use distributions of the values of the relevant parameters both over time within a run and between runs. We assume here that use of the average values of parameters (i.e., scalars) over the entire execution of the program provides enough information for good decisions about representation choice.

In other words, we are using a crude estimate of computation cost. This estimate should be useful for gross distinctions, but not fine ones. A human storage structure designer works much the same way: he often uses rough estimates of relative frequencies of operations and of relative sizes of data structure components to guide the choice of a storage structure.

Because we are not making precise distinctions for estimating cost, it is not necessary to formulate precise characterizations of the associative search techniques and storage structures in the library. It is adequate to outline each algorithm and compose a cost formula for it in terms of the costs of its primitive steps, the parameters that describe average data structure size, and the parameters that describe expected program behavior. After enough experience with automatic selection systems like this one, we will learn where (heuristics, libraries, formulas, parameters, etc.) additional effort should go.

After both space and time costs are estimated, if a representation is at least as expensive in both time and space as one already analyzed, it is eliminated as a candidate. Otherwise, its estimated overall computation cost is computed (see the discussion below), and it is added to an ordered list of candidates that have already been analyzed. The list is ordered by estimated overall computation cost. The final step in the current selection system is to print out detailed descriptions of the top few candidate representations on the list.

The choice for real programs of an appropriate overall measure of computation cost is a good research problem. Attempts at a solution might well lead to the development of new ways of describing programs and their behavior, and to new ideas and standards for the systems in which programs function. Such a study is certainly worthwhile, but although we could make use of the results, it is outside the scope of this work and is left for future research. Instead (following Reference 4), we have provided in the selection system a function which defines the overall computation cost of a candidate representation to be the product of its storage space and execution time requirements. It would be easy to define a new function of space and time to replace

the one provided, or to experiment with several in the context of the selection system.

Once cost estimates are derived for all candidates, we rely on the methods of Reference 4 to choose the best representation for the class of triples in the context of the entire program. In particular, we refer the reader to Low's book for a discussion of the following problems:

1. estimating and considering the cost of the non-associative operations of the program.
2. defining an "overall" cost function of space and time.
3. considering the interaction between representation choices for separate data structures whose life-times overlap.

RESULTS FOR THE SIMPLE EXAMPLE (Figure 2)

Based on its analysis and on the answers to questions, the system proposes and evaluates several possible storage structures for the Sexof relation. The best of these is a record having one field which points to the set of MALES (this set is represented as a linked list). Based on the cost formulas, this has an expected size of 5.5 cells and an expected time of 457 instructions. Another alternative would be to list each association separately in a table. This has an expected size of 13.0 and an expected time of 935. The set representation is better than this (and all others) in both time and space, and is chosen. This is a very simplified (almost degenerate) case of the selection system's operation.

In selecting a storage representation for the class based on the Parentof relation, the system goes through a similar procedure. In this case, it selects to assign a record for each person, with a field of the record indicating the set of parents.

Now let us consider the more complex program including the statement in the box. Since the Homeof relation is used, there are now three classes of associations. In our tiny sample program, the only use of Homeof is to test if someone lives in Rochester. This, plus the fact that there is a small fixed number of people, causes the selection system to choose as the best structure a single word per person indicating whether he lives in Rochester. This representation is estimated to require one cell and 300 instructions.

A more interesting effect of adding the boxed statement is to change the choice for representing the class based on the Parentof relation. Now the program calls for enumerating all Child-Parent pairs as well as finding the PARENT of a given child. The selection system considers making two separate data structures or making one data structure which does both jobs. The best choice is to have a list of all children with each child indicating the set of his parents. This is estimated to require 16.5 cells and 1600 instructions.

SUMMARY

This paper explores some of the problems of automatic representation selection for associative data structures. In particular, it presents methods for selecting in-core storage structures for programs which use associative data and operations which are similar to the ones that the SAIL programming language provides.

A model of the associative data and operations of such a program is defined. We describe a library of storage structure representations for three-part associations (triples) that includes records, property lists, inverted lists and hash tables. We show how to use the model in conjunction with the library to find "efficient" candidate representations for the associative data of a given program. This choice of candidate representations is guided by heuristic rules that embody educated assumptions about computation cost tradeoffs. In addition to representations that provide a single method of access to associative data, we consider representations that provide multiple access paths to the data and ones that provide multiple copies of the data.

Finally, we discuss methods for estimating the computation cost of each candidate representation, and for selecting the "best" one.

ACKNOWLEDGMENTS

The author is indebted to Jim Low, Jerry Feldman, and Tom Cheatham for their ideas, guidance, and encouragement during the course of this work, and to Peggy Meeker, who put up with my drafts, changes and deadlines cheerfully while doing a meticulous job of editing and typing the manuscript.

REFERENCES

1. McCarthy, J. et al., *LISP 1.5 Programmer's Manual*, M.I.T. Press, 1962.
2. Feldman, J. A. and P. D. Rovner, "An Algol-Based Associative Language," *CACM*, August 1969.
3. Rovner, P. D. and J. A. Feldman, "The LEAP Language and Data Structure," *IFIP Congress Proceedings*, 1968.
4. Low, J. R., *Automatic Coding: Choice of Data Structures*, ISR16, Birkhauser-Verlag, 1976.
5. Maurer, W. D. and T. G. Lewis, "Hash Table Methods," *ACM Computing Surveys*, Vol. 7, No. 1, 1975.
6. Rovner, P. D., "Automatic Representation Selection for Associative Data Structures," Ph.D. Thesis, Harvard University; also TR 10, Computer Science Department, University of Rochester, 1976.
7. Reboh, R. and E. Sacerdoti, "A Preliminary QLISP Manual," SRI AI Center Technical Note 81, August 1973.

Efficiency estimation—Controlling search in program synthesis

by ELAINE KANT
Stanford University
Stanford, California

People should be able to specify a program in a high level language and let the computer take care of finding an efficient target language implementation. One way to do this is by program synthesis. LIBRA is a system designed to guide the application of synthesis rules in the PSI program synthesis system. In the PSI system, there is a structured programming style of organization of programming knowledge. High level concepts such as sets may go through an arbitrary number of levels of refinement before target language concepts such as lists or arrays are produced. Refinement rules make the transformations between these concepts. If given such a set of refinement rules, LIBRA can guide the transformation of a high level program specification to a target language program. The target program is made efficient in two ways: by clever choice of refinements and by the application of optimizing transformations at any levels that are appropriate.

LIBRA uses the technique of efficiency estimation to guide a heuristic tree search of some of the possible refinement sequences. Efficiency estimation takes into consideration: a user-specified program cost evaluation function, resources allowed for program writing, data structure sizes, and conditional branching probabilities. An estimate of the cost of the program descriptions in the tree is maintained for comparison between alternatives, for identification of the most important decisions, and to trade off final program performance with the cost of writing the program. To simplify the decision making process, decisions are grouped into related sets or cost-independent blocks. Heuristic knowledge about efficient implementations for certain circumstances is also integrated into the system. A version of LIBRA has been written in INTERLISP; it has been used to guide the implementation of several LISP programs.

This work is described in more detail in *The Selection of Efficient Implementations for a High-Level Language* in **Proceedings of the Symposium on Artificial Intelligence and Programming Languages,** Rochester, New York, August 1977.

Transformational implementation

by DAVID WILE and ROBERT BALZER

Information Sciences Institute
Marina del Rey, California

Only modest gains in programming productivity have been produced in 25 years of software research, but the groundwork has been laid for major advances through rationalization and automated aids. This groundwork rests on two critical ideas: that specification must be separated from implementation, and that the separation between these two processes should be a formal operational abstract (i.e., very high level) program rather than a nonoperational requirements specification. Structured programming represents the first results of combining these ideas. It is a special case of a more general two-phase process, called Abstract Programming, in which an informal and imprecise specification is transformed into a formal abstract operational program, which is then transformed into a concrete (i.e., detailed low-level) program by optimization. Abstract Programming thus consists of a specification phase and an implementation (optimization) phase which share a formal abstract operational program as their common interface.

The concept of Abstract Programming is completed by adding the feedback loops required by Testing, Maintenance, and Tuning. In conventional programming where no abstract program exists, these feedback loops all operate on the optimized concrete program. On the other hand, in Abstract Programming, if an effective method can be found for guaranteeing the validity of an implementation (that is, the functional equivalence of the abstract and concrete programs), then the validation process can be shifted to the specification phase to show equivalence between the user requirements and the abstract program. Thus, validation could, and should, occur before any implementation. Furthermore, if the implementation process could be made inexpensive through computer aids, then maintenance could be performed by modifying the specification and reimplementing it rather than directly modifying the optimized concrete program, as is current practice. The importance of such an advance can be recognized when one realizes that optimization is the process of maximally spreading information (to remove redundant processing), and that modification requires information localization. Thus, the two processes are diametrically opposed; this fact explains much of the current problem with modifying and maintaining existing programs. The second major cause of this dilemma is that optimization obscures clarity and thus makes it difficult for maintainers even to understand how the concrete program operates.

It is therefore clear that major advances in programming will hinge on the ability to provide an inexpensive optimization process with guaranteed validity so that maintenance and validation can occur in the specification phase on the abstract program rather than in the implementation phase on the concrete program.

Several researchers have espoused an approach which we will call Transformational Implementation (TI), in which equivalence-preserving transformations are successively applied to the abstract program to effect such an optimization. The key to this approach is that while optimizing transformations are selected by the programmer, the program is transformed by the computer system. Hence, the programmer designs the optimization; the machine ensures its validity and transforms the program.

In addition to guaranteeing the validity of the implementation, drastically reducing the time and effort required to implement a system, and enabling maintenance to be performed at the conceptual level on the abstract program, this approach would also allow programmers to experiment with alternative implementations to widen their experience base and improve their capacity to design optimizations.

We have undertaken a study to discover: what facilities are necessary for the TI approach; what transformations are required; how are they expressed; what categories do they fall into; how can they be named; what kinds of applicability criteria are required; what processes are required to verify that these criteria are satisfied; what instrumentation facilities are required to test a transformation's selection criteria; what proof techniques are required to validate the transformation; etc. In short, the study will provide an experience base for the TI approach from which we can design and implement a prototype system.

REFERENCE

1. Balzer, Robert, Neil Goldman, and David Wile, "On the Transformational Implementation Approach to Programming," *2nd International Conference on Software Engineering*, October 1976, p. 337.

A panel session—Whither automatic programming

SESSION CHAIRMAN—ROBERT BALZER
Information Sciences Institute

Panel Members

Thomas S. Standish—University of California
Michael Hammer—MIT Laboratory for Computer Science

PANEL OVERVIEW—Robert Balzer

The preceding sessions have displayed some current research into the area of Automatic Programming. These efforts are small laboratory prototypes which attempt to automate particular portions of the programming process. It is quite clear that to progress significantly beyond our current state of software development technology, increased automation, in some form, must occur.

The questions facing this panel, and addressed below, are when, in what form, and to what extent this automation will occur?

THE FUTURE OF AUTOMATIC PROGRAMMING—Thomas A. Standish

In *automatic programming,* we strive to build systems that will perform problem acquisition, algorithm and data representation synthesis, optimization, and concrete program generation—systems which automatically acquire specifications of the program behavior required and which will build programs to satisfy them.

I speculate that the attainment of technological capability to build such automatic programming systems flexibly across a spectrum of dissimilar, important problem domains is a long way off—perhaps centuries and most likely decades. Before we succeed, we shall likely have had to develop deep understanding of the interrelation between knowledge systems of many sorts—knowledge systems in which problems are posed and in whose terms solutions must be delivered (the so-called problem domains), knowledge systems describing the media in which problem solving computations must execute at the lowest levels (the so-called concrete solution domains), and knowledge systems that cooperate together to provide the scaffolding in which program synthesis can take place (the intermediate problem-solving knowledge systems, which offer systems of reasoning and representation).

The exploration required to develop an understanding of relationships between these several sorts of knowledge systems is likely to lead to a deep reorganization of the foundations of epistemology itself. For example, the *doctrine of reductionism*—which holds that phenomena at higher levels of science can be explained by reduction to laws and facts at lower levels—is a doctrine which is far too simplistic and inadequate to provide understanding of how to build program synthesizing systems. *Reductionism* may describe the end *product* of program synthesis—i.e., the already synthesized program—but it seems inadequate to describe the *process* of synthesis itself.

The doctrine of reductionism is important in programming. Every large software system is living testimony to the idea of reductionism. In such systems, we organize the data and operations at given levels to represent the information and problem solving processes required at higher levels. When we engage in "top-down programming" or "programming by stepwise refinement," we consciously organize the act of large system construction into a number of levels of reductionistic choice, each carefully selected to be intellectually manageable and to permit flexibility in future maintenance.

While every synthesized software system is an example of successful reductionism in which the external high-level behaviors of the system have been reduced to a composition of the lowest level behaviors of the underlying machine, there may be a *paradox.* The knowledge systems in the heads of programmers who build the overall system may not be similarly reducible. Rather, such knowledge systems may exhibit phenomena akin to the logical independence of the parallel postulate from the remaining axioms of Euclidean geometry.

The history of science teaches us that large increments in technological power are seldom purchased through cheap tricks or shallow understanding. Rather, deep basic understanding of a degree of indirectness scarcely imagined by original explorers seems more characteristic (recall here the history of the conquest of flight, or the history of the quest to transmute the elements).

It seems to me that basic understanding of how to represent knowledge systems, and how to use one knowledge system to represent and solve problems posed in another, is a basic requirement of the task of building program synthesis systems. Perhaps it is better to make an indirect assault

on deepening our knowledge in these areas before making direct assaults on the synthesis of particular categories of programs. After all, the Wright Brothers built a wind tunnel and studied properties of airfoils extensively before they conquered flight.

THE IMPACT OF AUTOMATIC PROGRAMMING RESEARCH—Michael Hammer

Contemporary automatic programming research encompasses a variety of approaches to the goal of transferring some of the programming function from human to machine and thereby alleviating the software problem. A very high level language system translates a program expressed in terms of nonprocedural, problem-oriented constructs into conventional algorithmic code; in so doing, it must perform global program transformations in order to achieve an efficient object program. A knowledge-based system uses the semantics of a particular problem domain to interpret a user's functional specifications for a program in that domain and to construct a system with the specified functionality. A program synthesis system utilizes formal theorem-proving techniques to transform non-functional program specifications (typically expressed in terms of a set of input-output pairs) into an executable program.

Although extensive research has been conducted in each of these areas, and significant advances have been made in addressing the technical problems that each presents, it is unlikely that any of this work will culminate in systems that will meet the ambitious goals that have been set for them. The very high level languages that have been developed so far are adequate for expressing the main body of a program in a nonprocedural way; however, they do not address issues related to deviations from the normal flow of the computation, such as special case recognition, exception handling, and coping with erroneous data or system failure. Furthermore, introducing a new language into an operational environment has in the past proven to be a major undertaking, one that raises serious managerial and technical problems. The principal problem facing the more advanced, artificial intelligence systems is one of scale. It is doubtful that the techniques that suffice for laboratory prototypes can be extended to cope with realistically rich problem domains and large, complex programs.

The difficulty underlying all of these problems is that the fundamental premise of automatic programming is simply false. This premise is that, after thirty years' experience in writing programs, we now know enough about the programming process to automate it. On the contrary, it appears that our understanding of the nature of programming is weak. The ad hoc attacks on the software problem that are currently feasible are not likely to produce major results.

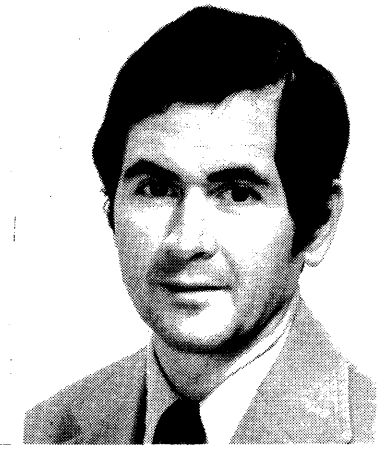
Nonetheless, automatic programming research will have important consequences, both direct and indirect, for the future of software construction. In the near term, we should see limited systems that automate one part of the programming process or that are adequate for the synthesis of small programs; in either case, the context will be a simple problem domain that has a well-defined and understood semantic structure. Examples of such systems include an automatic data base design system and a facility for generating simulation studies. Another possibility is a system that uses knowledge of an application domain to assist a user in deriving a complete and consistent set of program specifications. In another vein, very high level language research should lead to the design of formal specification languages for effective and precise inter-personal communication.

However, the most significant benefit of automatic programming research should be a deeper understanding of programming by human programmers. It has often been the case that the attempt to automate an activity has forced its practitioners to understand and systematize their actions; automatic programming research should have the same effect on programming. In the attempt to computerize the software production process, it is uncovering important general programming principles and techniques. The structures and guidelines that derive from this work will enable and encourage programmers to be more disciplined and organized; in the long run, they can contribute to turning programming from mysticism into science. In other words, automatic programming research should benefit software engineering practice.

PART III—SYSTEMS

Professor Norman Abramson

Area Co-Director:
Norman Abramson
University of Hawaii
Honolulu, Hawaii



Area Co-Director
Eugene R. Cacciamani, Jr.
American Satellite Corporation
Germantown, Maryland

Data networks

In this area we explore the technology and development of information networks on a national and international basis. There are four sessions in this area and they deal with (1) "The Architecture and Application of Nationwide Packet Switching Networks," (2) "International Computer Communications," (3) "International Public Data Networks," and, (4) "Satellite Data Communications for Public Service."

The first session focuses upon the architectural similarities and differences of several packet-type networks and how the application of these networks influences network architecture which have recently become operational or are under development.

The first of two sessions on international computer networks will cover matching future international regulations against the continued growth of computer communications systems and technology utilizing all types of international transmission media. The concerns of this session include the non-homogeneous nature of national regulations as they apply to international computer communications systems, transborder data flow, and the need for new regulatory considerations for international satellite-oriented computer communications networks.

The second international session emphasizes the operational problems of international data networks with multiple organizations interconnecting dissimilar equipment rather than a single, coordinated, homogeneous network. This inhomogeneity creates special problems and requirements in order to create a viable service. This session will describe some aspects of the interconnection problem, potential solutions, and some internetworking field trials underway.

The final session will deal with the emerging question of information networks for public service users in the United States. These users consist of a small number of related communities—libraries, educational institutions, health centers, and local government agencies. Each of these communities has a need for data communication capabilities with certain common characteristics. Among these characteristics are a high degree of connectivity within a highly dispersed user community, the ability to distribute relatively small amounts of data with high economic or social value and a modular structure which can accommodate rapidly changing and diverse data rates within the same system. The key to

satisfying these data communication needs is the aggregation of the public service data communications market by means of communications satellites and the broadcast data communications architecture they make possible. In this session we describe the projected requirements of this aggregated market and discuss the status of current proposals to provide these data services.

Challenges in the planning of international communications

by DAVID J. HORTON
Hawaiian Telephone Company
Honolulu, Hawaii

It's difficult sometimes to realize how far international telecommunication has come in 20 short years. As recently as 1957, "Hawaii Calls" was broadcast using HF radio. After the program was switched to transpacific cable, simulated fade had to be added to recapture the flavor of "music from the isles." Today we've progressed to the point where we can provide a small store on Molokai with on-line data communications to a data base in Paris—at low cost and virtually error free.

I'd like to explore with you the environment under which all this achievement came about and to examine some real areas of concern which may inhibit continued progress. While some of my remarks may apply generally, my emphasis will be on data communications in the Pacific Basin.

I have a very simple definition of planning: A process which results in spending the right amount of money, on the right amount and right kind of equipment at the right time. The definition is simple; the process, alas, is always difficult and rarely perfect. Planning in the face of rapid technological change and everchanging market needs is particularly tricky. When the international dimension is added, the complexity reaches quite awesome proportions. Yet, decisions must be made or opportunities will be lost and/or needed services will not be provided.

Let's look at some of the elements that enter the picture when we go international:

- Apart from the obvious problems of language and cultural differences, we must recognize the different ways that telecommunication is viewed in different countries. In North America for example, we aim for the lowest possible cost consistent with good service and a fair rate of return. Most other countries are inclined to view telecommunication as a source of high profit with which to subsidize some of the weaker operations, most notably the postal service.
- Political considerations cannot be ignored. Here we must be concerned not only with political ambition, but with the opportunities of political progress. We can contemplate, for example, the phenomenal increase in communications which would accrue from full trade with the People's Republic of China.
- We must concern ourselves quite deeply with some of the reservations being expressed regarding the presence of data bases in foreign countries. Not only is this viewed as an unnecessary import, but there is very valid worry regarding what is left of our privacy.
- Markets are different as we travel around the world. We find different terminal types, different protocols, and different speeds—happily, however, major success is being achieved in the area of international standards. I can think of no area more deserving of our continued attention than this. We, as carriers, have a profound obligation to ensure a high degree of uniformity in the networks we design for Spaceship Earth. We have the skills, and we have the vehicle (CCITT)—we will have no excuses for failure.
- Geography differs widely, as is obvious when one compares the wide open spaces of Canada with the density of population in the Netherlands.
- In satisfying customer needs with new equipment, it is frequently not enough that it was invented "here", but it also must be designed here, developed here, manufactured here, sold here, and serviced here. And, often the customer must wait until these national objectives can be satisfied.
- And finally, planners must contend with all of the unpredictability inherent in the presence of extensive competition for international telecommunications.

But, complex as it may be, planning must continue. Good planning is essential and plans, once agreed, should meet time and cost targets or both carriers and customers will suffer. This is particularly true of international planning where the inability of planning members to fulfill their commitments can cause considerable difficulty for their correspondents.

International planning in telecommunications has had its problems to date but has nothing to be ashamed of. It is interesting to recall that the first major achievement was the laying of the telegraph cable between two countries which today provide the butt of many a joke; namely, Newfoundland and Ireland—and that this occurred in the year 1856, 20 years before the invention of the telephone.

Progress since then has been quite rapid (with most of it coming within the last 15-20 years) to the point where we now have extensive cable and satellite facilities encircling the globe.

At the 6th Annual International Conference on Planning, held last fall in Honolulu, we were treated to paper after

paper about the growth in the Pacific Basin with special emphasis on the emergence of Korea, Taiwan, Singapore, and Hong Kong as "new Japans" with all that it implies in terms of increased trade and thus, telecommunication traffic. It is obvious that good planning is essential if we are to meet the emerging needs of customers in these Pacific Rim countries.

Thus far, we've gone through a progression of complexity: fast moving technology; increased and changing market demands; and a high requirement for international cooperation. We must now add yet another challenge—international *data* communications.

The past few years have seen explosive growth in international data communications and with the advent of packet switching, this growth will continue at an ever-increasing pace. This latest technological wonder which has been with us a very short time finds its roots in the ARPA network but its branches now encircle the world. At last count, there were almost 20 such networks in service or about to be in service.

Packet switching respects no boundaries. It was internationally almost before it became domestically available. The Atlantic was the first pond to be hurdled, and we have today extensive access to North American data bases by European customers. Similar plans are already well under way in the Pacific area.

Technologically, packet switching is a hyperactive child, constantly exploring new possibilities—and now and again getting itself in a little trouble. The pace of technological change is higher than in any other area of telecommunications and with the introduction of microprocessors, we find ourselves using third generation technology within three years of the first network going into commercial service.

Data communications customers are a demanding breed—demanding that new technology be introduced fast, demanding shorter time frames than they have traditionally been promised, demanding the features that they need, and perhaps most importantly, attaching a very valid threat of taking their business elsewhere (or doing it themselves) if these demands are not met.

There is of course much to be done, and I've touched on some of the tasks:

- harnessing the right technology to meet real market needs
- standards
- international dialogue

One area, however, which could become a major stumbling block is not within the carriers' control. I'm speaking, of course, of regulation, or more precisely, the regulatory process.

Let me suggest that regulation is here to stay and that it will probably get worse before it gets better.

In a December 1977 Time magazine, President Carter was quoted as follows: "Regulations should be as simple and clear as possible. They should achieve legislative goals effectively and efficiently. They should not impose unnecessary burdens on the economy, on individuals, on public or private organizations, or on state and local governments."

How about that? That was in December 1977. You all notice the big improvement?

To be fair, the problems are not entirely the fault of the FCC. Once the decision is made to have controlled competition as opposed to a monopoly, it follows logically that there will be a greater workload. It is the FCC's inability to handle this workload that causes so much pain, not only domestically, but also with our partners in other lands. At a meeting of Pacific cable operators held in Honolulu at the Kahala Hilton Hotel in September 1977, all countries expressed dismay at the regulatory lag caused by the FCC. In one cited example, a submarine cable cut into service one and one-half years late and cost $40 million over budget as a result of this delay.

And, if I may use an example much closer to my own experience, let me tell you that from the start of the initial discussion with Telenet until the installation of a working node in Honolulu, less than three months expired. More specifically, from the placing of the order to the acceptance of installation was a mere six weeks, or more specifically, still, from the delivery of equipment to turn up (capable of providing service) five working days expired. By comparison, our request for approval to construct was filed on August 12th, 1977 and approved about December 15th with an effective date of January 3rd, 1978. I think you can imagine the disappointment that is caused to carrier and customer alike as a willing switch capable of earning revenue sits gathering dust as the paper flows around Washington. This is simply not adequate regardless of what the problems and what the excuses are.

The fact is that whether we like it or not (and I personally do) technology is moving fast and market demands are increasing daily, resulting in the need for fast response from carriers both domestically and internationally. The regulatory process must keep up.

How do we do this? I wish I knew.

I don't believe that simply increasing the staff at the FCC is the answer. So let me offer some suggestions which may be of help:

1. Establish guidelines and policies for international telecommunication.
2. Clarify the roles of individual carriers if in fact there are any valid reasons for segmenting the market among them.
3. Anticipate the effect of technological likelihoods, and
4. Based on carrier-provided information, anticipate the kind of action that market demands will require not only of the carrier, but of the regulatory process itself.

Let me give you an example:

We're now in the middle of an extremely costly exercise called Computer Inquiry II. There's a great deal at stake and some very vocal and widely different views are being expressed by various lobbies. What bothers me most about this is that it's only six years since we were presented with the wisdom we have come to know and love as *the* computer

inquiry. But it wasn't *the* computer inquiry—it lasted a mere six years. Why? Mainly because its authors failed to understand where technology and the demands of the market place were taking us. And at the very moment in history when everyone finally realized the line between communications and data processing was a broad blur, the FCC presents us with a definition which claims to draw a major distinction. It's thus difficult to be optimistic that CI2 will be a big improvement on its predecessor.

Let me stress again that we must have a thorough understanding of what technology can do, what the customer wants, and what the various vendors are trying to achieve. We must then produce regulations that are fair, of course, but which stimulate innovation and keep prices as low as possible. A difficult task, indeed, but one which must be undertaken. I believe the carriers themselves should play an active role in trying to streamline regulation. I would hope that both they and the FCC would welcome this.

The need for continuation of full-period transparent private-line service

by PHILLIP C. ONSTAD
Data Services, Control Data Corporation
Greenwich, Connecticut

During the past year several moves have been made by the international telecommunications industry which may threaten the continuation of the flexibility currently available to the user of telecommunications services for data transmission. The major problem now confronting the user of international telecommunications services for data transmission is the hinted or announced intention of the international and foreign carriers to offer "value added" services as a replacement rather than an alternative to full-period transparent private-line services. The availability of transparent private circuits permits technological innovation in signaling and in information processing network development to the benefit of all. Loss of transparent, flat-rate, private circuits would retard advances in distributed processing and shared data base developments. Furthermore, many existing teleprocessing systems would be degraded in effectiveness and in many cases their data services would be withdrawn.

With the advancement of computer technology during the fifties and sixties, it became evident that the full potential of data processing and information systems would not be forthcoming until a faster and more efficient means of transportation could be developed to move information in and out of the data processing systems. While data processing systems had the power to manipulate information at split second speeds and store vast amounts of data which could be retrieved in fractions of seconds, these capabilities were of little use unless the user was within close proximity to the data processing system. Transportation by means of mail or delivery services worked well for batch processing operations such as payroll, sales analysis, and other operations where there was no need for an immediate answer to a problem. However, these transportation methods could not satisfy the demands of applications such as reservation systems or production control systems where information was needed on an immediate basis.

The obvious solution to this problem lay in the use of the vast telecommunications networks covering the country and the world. Although these networks operated in an analog mode and were used for transmission of voice and low speed telegraph signals, equipment was developed for connection between the data processing systems and terminals and the telecommunications network which could convert the signals from digital to analog and then back to digital at the other end. Thus began the era of telecommunications dependent data processing systems.

In the early days of data transmission, the speed of transmission was limited by many factors and the reliability of the transmission was low. In addition, there was very little interest by the communications industry, whose major purpose was to provide voice and low speed message communications service, and not to provide methods for the high speed data transmission and high reliability factors required by the data processing systems. Thus, it was left to the data processing and related industries to develop the necessary equipment and procedures.

The telecommunications common carriers offer transparent telecommunications circuits on both the private-line and switched networks; these are available in various speeds. By utilizing these circuits, the data processing industry developed or perfected equipment and techniques which made the transparent communications facility a highly efficient and highly reliable method of transportation of information from remotely located input/output devices to and from the data processing system.

Once this means of transportation was proved feasible, the number of telecommunications-oriented data processing applications exploded. This demanded increasing technological development to obtain optimum performance of the transportation medium in order to maintain optimum performance of the overall data processing system.

The data processing industry has been and continues to be a highly competitive and highly complex industry with many companies manufacturing and marketing many different types of equipment and systems from the very small micro processors to huge information handling systems. To remain competitive in the marketplace each company has developed unique architecture, technologies, coding methods, circuit logic, storage methods, protocols, etc., in order to make their systems more reliable and faster than others. Each system has a specific balance of hardware and software control to maintain optimum performance from a specific system and specific application. In maintaining this balance, the user must also have control over all input and output functions, including remote terminal devices that carry information to and from the system.

When preparing a new application for a data processing

system, the programmer and system designer must have the flexibility to utilize the best combination of computer logic, storage media, input/output methods and devices. Further design must be made to the transportation system from remote terminal devices to the processor in such a way that information is received and released in a manner which meets the requirements of the application, such as formats for messages, etc. It must also meet the requirements of the data processing system to properly mesh these input/output operations with the multitude of other operations going on during the processing of a major application. The importance of the input/output operations increases as the complexity of the application increases. The effect of several low speed terminals coming into a system or application which is primarily batch oriented is minimal and does not require a great deal of control over the input/output method. However, a system operating a large reservation or time sharing application involving hundreds of terminals requires a highly complex input/output protocol with almost total control over the telecommunications activity in order to maintain optimum system performance.

The transparent nature of the telecommunications networks has given the application designer the flexibility to design the transmission protocol and utilize equipment which best meets the requirements of the overall system environment. The designer may choose the speed of transmission, the line protocol to be used, the coding structure, block lengths, error correction methods, retransmission methods, etc. He has the flexibility to utilize the usage-sensitive switched circuits of the public networks or full-period line service. He also has the flexibility to utilize asynchronous or synchronous transmission methods and may utilize various types of equipment such as multiplexors, concentrators etc., to obtain maximum utilization from the telecommunications service. This flexibility is extremely important to the operation of a major telecommunications dependent data processing application. Hundreds of data processing systems of this type are operating throughout the world today. Many of these systems require thousands of miles of private lines which are leased from the common carriers, both nationally and internationally. They also connect to the public-swtiched network for many applications.

Several years ago a new method of data transmission was introduced with the advent of the "value added" carriers. This group of regulated carriers lease full period private-line services from other carriers and, through the use of various types of telecommunications and data processing equipment and software in conjunction with the private line services, establish a stand alone data transmission network with a protocol *best suited to the operation of the network*. This "valued added" service then is offered to users who do not wish to establish and operate their own telecommunications networks but have some telecommunications requirements in their data processing applications.

Although these "value added" data transmission services do fulfill the telecommunications network requirements of many users operating data processing systems with small- or medium-scale remote terminal operations and further allow the user to share a network with other users, in most cases they are of little value to the user with a large telecommunications network requirement. This is because they have been designed with the optimum performance of the network rather than the optimum performance of any single application as the criterion. In utilizing these networks, the user must conform to the protocol requirements of the "value added network," therefore losing much of the flexibility available in the transparent network. This is not to say that the "value added networks" do not furnish a valuable service to the small- and medium-scale user on a national basis. Indeed, the savings in telecommunications cost available through use of this type of service is in many cases such that they provide the economic feasibility of performing some telecommunications functions in an application normally run in batch mode.

In the area of international data transmission, the value added services provide the only means of communications on a *usage sensitive basis* between the United States and foreign points. The only other alternatives available to the user are the very slow telex service at high cost or the use of full period international private-line service, which, because of its extremely high cost, is only economically feasible in the cases of very large requirements or with users needing an alternative voice data capability which might offset the high cost. In many cases even the large user will find it more feasible to utilize the value added type service to points having small transmission requirements even though response and/or system optimization may suffer somewhat when adapting to the protocol of the value added service.

To summarize, the intelligent networks provided by the value added services will fulfill the requirements of a great number of telecommunications-dependent data processing applications and systems. However, the large variety of data processing systems, each operating under a unique architecture and the various requirements of the large telecommunications dependent data processing applications, demand that the user have complete flexibility to utilize the best data transmission method possible. This is true whether those methods be transparent or value added service, private-line or switched network service, in order to design and operate the optimum environment for the data processing applications.

During the past year several moves have been made by the international telecommunications industry which may threaten the continuation of the flexibility currently available to the user of telecommunications services for data transmission.

As previously stated, the new value added type services make available a viable alternative to fulfill the needs of a significant number of telecommunications users. However, these type services have serious drawbacks and cannot be applied to all user requirements. They are also relatively new and have yet to be totally proven from economic as well as reliability standpoints. While the growth of these services has been somewhat slow within the United States with only three independent carriers currently offering the service, the international common carriers, as well as a number of foreign telecommunications administrations are

moving ahead at a substantial pace to have these value added services in place as soon as possible. This has led to additional problems to the users as the fast pace toward installation of these services has not allowed the proper consideration for much technical standardization among the various services, thus creating a number of potential problems for the user.

The technical problems leading toward some type of standardization of these networks is currently being handled by the Study Group VII of the Consultive Committee on International Telegraph and Telephone of the International Telecommunications Union. Although this group has made significant progress toward solving of the standardization of the protocols and other technical criteria which will allow for the eventual interconnection of these networks, much work must still be done before proper solutions are found to make these networks truly viable on an international basis.

The major problem now confronting the user of international telecommunications services for data transmission is the hinted or announced intention of the international and foreign carrier to offer these new services as a *replacement* rather than an alternative to full period transparent private line services.

The possible replacement of full period transparent private line services for international and foreign communications has been brought to the attention of users during the past year by:

- The introduction of a contribution by the Italian delegation to the CCITT Study Group III on Rates and Tariff Matters requesting the group to study the feasibility of replacing full-period private-line services with some method of usage-sensitive or volume-sensitive pricing for the service.
- The refusal of an internaional common carrier to furnish full-period private line service to the United States. The refusal is based on the availability of a new usage-sensitive service between that point and the United States which would now fulfill prospective customers' requirements.
- Articles now appearing in the Japanese press which indicate that Japan plans to have their new Venus Value Added Service operating within the next year or so to replace for private line services for data transmission.

Solutions to these problems which are of true benefit to the user are complex and difficult to obtain. With the exception of the United States Federal Communications Commission, which is charged with the protection of the public interest in the area of telecommunications, there is very little opportunity to present a user's point of view in international and foreign telecommunications services.

Since the only international body having an interest in this area is the Consultive Committee on International Telegraph and Telephone of the International Telecommunications Union, it is of critical importance that users take an active interest in the activities of the American delegation to the CCITT.

In view of the pattern of rapid technological advancement and economic growth in the data processing and telecommunications industries, we believe that a study of the tariff system for full period private line service may be well advised. Because the tariff system profoundly affects the utilization and development of communications based data processing systems as well as other communications based systems, we believe that such a study, if properly handled and provided it receives input from the user community as well as the carriers, may result in workable solutions to the issues at hand. However, such a study must be conducted with a very open mind and without preconceived notions on the part of the international and foreign carriers that replacement of full period transparent private line services is the only solution.

There can be no doubt that the world has entered the information age where continued growth of all elements of society are mutually dependent on the rapid flow of information with the minimum of hindrances. The EDP industry has spearheaded the advancement of information transfer through its technological and manufacturing advances. Prominent among those applying these advances to the direct benefit of the public are the financial, retail and health care activities and certainly the telecommunications administrations.

A key factor in the technological advance of ever-increasing telecommunications oriented data processing has been the widespread availability of full-period point-to-point and multipoint private leased circuits. The most important characteristics of privately leased circuits have been their transparency and a tariff system independent of the volume of information transferrred. Transparency as used in this context denotes a point-to-point or multipoint analog circuit which delivers the electrical signals applied by the user with the practical minimum of delay and distortion consistent with the applicable tariffed characteristics.

The availability of transparent private circuits permits technological innovation in signaling and in information processing network development to the benefit of all concerned. Presently, these innovations can proceed observing only the electrical interface without the constraints of all establishments and telecommunications network administrative control protocols and without the potentially excessive delays even in virtual circuits.

There is no doubt that the introduction by the administrations of new services will stimulate the growth of new information systems which would not have emerged otherwise. There is no essential conflict between the new services and privately leased circuits and it would be contrary to every interest for any tariff system to be based on an unfounded fear of competition.

Loss of transparent flat-rate private circuits would indisputably retard advances in distributed processing and shared data base developments. Furthermore, many existing teleprocessing systems would be degraded in effectiveness and in many cases their services would be withdrawn. Additionally, and as a direct consequence, administrations would suffer erosion of their revenues rather than safeguarding their revenue.

Satellite business systems—Innovative services for business communications

by RONALD W. McCABE
Satellite Business Systems
McLean, Virginia

INTRODUCTION

It is a pleasure to participate in the National Computer Conference, and to represent SBS in this session on International Computer Communications. The topic is a most timely one for SBS, even though we are a domestic carrier, because during our extensive market investigations the desire of potential customers to improve their ability to connect their domestic and foreign data transmission capabilities has been expressed quite strongly. SBS believes its proposed offering will provide customers a significantly improved digital network capability within the United States, and SBS is supportive of industry efforts such as this which recognize these expressed concerns of users and which seek to promote an improved environment for uniform international networks. But before I discuss the specific topic of today's meeting, I would like to give you some background on SBS and describe its plans and recent activities.

SBS proposes to offer innovative, high data rate, private switched communications networks for industrial and governmental users. These networks will be designed primarily for organizations requiring large communications networks that have heavy and dynamic traffic loads, and will provide integrated digital transmission of voice, data, and image traffic among geographically dispersed locations—much as a private branch exchange (PBX) provides switched communications within a single facility. Importantly, it is this same class of user which is more likely to be multinational in scope and to have far-reaching requirements for various types of international communications. Thus, SBS' market emphasis is closely allied with the matters of interest here today.

As you are probably aware, the SBS system will operate at higher frequencies (12/14 GHz) than present terrestrial and satellite carriers to minimize frequency interference with other systems. This will permit more geographical flexibility in the location of earth stations than is possible at the lower frequencies. These earth stations, which will incorporate specially-developed modulation and access equipment, will be located on the customer's premises, thereby minimizing the cost of access from customer nonearth-station locations to the nearest earth station customer location.

Satellite Business Systems is a partnership of COMSAT General Business Communications, Inc., Information Satellite Corporation, and Aetna Satellite Communications, Inc. These partners are, respectively, wholly-owned subsidiaries of COMSAT General Corporation, International Business Machines Corporation, and Aetna Life & Casualty Company.

SBS filed applications with the Federal Communications Commission in December 1975 and obtained approval for operation in February 1977. Major system developments and procurements are now under way, and we look forward to implementation. SBS expects to begin providing services with its operational system in early 1981. Prior to that time, SBS will be engaged in an intensive pre-operational program using leased satellite and terrestrial transmission capacity to connect a number of IBM locations; this activity is designed to evaluate the service approach and communications techniques planned in the operational system. The first phase of this effort, now successfully completed, involved experimental traffic tests between SBS earth stations at Poughkeepsie, New York and Los Gatos, California. Phase II has been under way since early this year and consists of a common carrier service, offered under terms of a tariff on file with the FCC, among these two locations and a third SBS earth station installed in Raleigh, North Carolina. An important objective of the Phase II program is to provide a stable environment into which components of the SBS operational system can be introduced and field trials conducted.

Another activity designed to demonstrate and better understand the usefulness of enhanced communications in the business environment was the PROJECT PRELUDE experiment which was recently completed. SBS was authorized by the FCC and NASA to use the Communications Technology Satellite, owned jointly by NASA and the Canadian Department of Communications, and transportable earth stations to conduct an innovative business communications experiment with newly-developed terminal equipment and business machines. The experiments were conducted successively at three pairs of business locations at the host companies—Rockwell International, Texaco, and Montgomery Ward. These companies were selected because they represented a cross section of the user community, and

helped to assure that representative data were collected. Experimental transmission of high speed data, high speed facsimile, and voice and televisual communications took place and the results were reported in the spring to both NASA and the FCC. In general, these experiments solidified our belief that these emerging applications will indeed satisfy unmet communications demands and become widespread in future networks.

SBS UNIQUE CHARACTERISTICS

Now let me describe briefly the uniqueness of the services that SBS plans by contrasting them with present communications. Satellite services are generally provided through very large earth stations located at some distance from urban centers because of their size, and to avoid interfering with terrestrial microwave facilities. Some small earth stations have been introduced but only in a limited way, again to avoid interfering with existing systems. The services currently offered by both terrestrial and satellite systems are primarily of an analog, full-time, point-to-point nature, and the switched services currently available are for voice and low-speed data.

In contrast to this environment, SBS will offer innovative, high data rate, private switched communications networks. These networks will provide integrated digital transmission of voice, data and image traffic among geographically dispersed locations, with full switching capability from one location to another.

Network is a key word in understanding SBS's uniqueness. It implies that there is connectivity among all of the locations at any time. This particular feature is perhaps the primary factor in SBS's system design. SBS will provide a customer complete connectivity among all his earth stations with sufficient transmission capacity to meet normal operating requirements; additional capacity will be available, on demand, to meet incremental requirements. This service approach, offered through small earth stations located on the customer's premises, results in a single multi-application network capable of handling virtually all of a customer's internal communications.

Locating earth stations on the customer's premises also provides a high level of security. This, combined with the use of time division, multiple access (TDMA), will make it impractical to intercept or decipher communications. The modulation and access equipment, burst modem, and radio frequency equipment of the earth station operate in a controlled access environment. Further, privacy equipment (such as cryptographic encoders) can be adapted for use as required.

SYSTEM COMPONENTS AND MANAGEMENT

To provide these dynamic, switched high data rate digital transmission services to the locations of an enterprise, SBS will install at each customer's traffic concentration points earth stations, owned and operated by SBS, through which he will obtain access to his network. The stations will use antennas five or seven meters (16 or 23 feet) in diameter and will include innovative modulation and access equipment to accomplish switching and multiplexing and to control the assignment of satellite capacity among the earth stations serving each customer. A variety of access ports is provided in each earth station for analog signals and for a broad spectrum of digital rates, based on the amount and types of traffic requirements of that particular station.

The system that SBS intends to implement initially will include three satellites—two in orbit (one spare), and one ground spare. As part of the SBS service, the customer will be able to obtain satellite transmission capacity for his private network on a full-period dedicated basis, and on an on-demand basis for overflow situations. The customer's total leased satellite transmission capacity will be dynamically allocated to meet his traffic demands (both as to routing and type—voice, data or image) based on priority assignments he establishes.

Within each customer's network, the SBS service will include a network management facility. This "window" into the customer's communications network will enable him to monitor network status and performance, change traffic handling priorities, order changes in service and collect usage statistics for internal accounting purposes.

A separate management facility will be used internally by SBS to monitor satellite and earth station performance, assist customers in designing networks to satisfy their requirements, distribute the customer networks efficiently among the satellite's transponders, and support system maintenance.

The proposed SBS system will best serve organizations requiring large communications networks with heavy and dynamic traffic loads. To install earth stations at all of a customer's remote sites, including those where the traffic would not justify an earth station, is not the intent. Rather, the objective is to provide an optimum system for each customer in which SBS ties together his traffic concentration points, while smaller offices—each connected by terrestrial services to the nearest earth station—are also brought closer to the center of the organization.

SBS has strived since its inception to look beyond the present needs of potential customers and to address, and hopefully provide for, their future requirements. The PROJECT PRELUDE experiments discussed previously touched on this future perspective by assessing new satellite applications in actual company operating environments. Additionally, SBS recognizes the absolute necessity for advanced terminal equipment to be available in the marketplace when SBS becomes operational. To assist in this, SBS has held a series of vendor conferences to advise about SBS system plans and capabilities, provide interface and service information, and to encourage these companies to intensify their development of advanced communications products. A conference for facsimile equipment manufacturers was held in December 1976 and a second conference for televisual equipment vendors was held in November 1977. A data

transmission equipment vendors' conference is planned for the near future.

ADVANCED APPLICATION POTENTIAL

It is clear that the SBS offering must provide capability for traditional voice and low speed data networks, but it is in the area of the emerging advanced applications that SBS sees the greatest future potential. SBS has conducted a number of in-depth case studies of the future communications requirements of individual potential customers and I would like to summarize for you briefly some of the findings. The advanced applications requirements we have identified and structured our proposed offering to serve would seem to be just as applicable in an international context as they are domestically. It should be emphasized that the leased services which SBS proposes to offer are for 24 hours, seven days per week. It is expected that this capacity, used during the day for conventional voice and low speed data communications, will allow for buffering of domestic batches of high speed data and facsimile for off hours shipment. And based on our investigations, an international requirement exists for this same kind of capability, taking advantage of off hours and time zone differences.

High-speed data communications

One clearly expressed requirement is for high-speed digital transmission, and SBS will offer data communications users a fully-switched network capability among a customer's earth station locations. Data rates include all standard speeds and range up to 6.3 megabits per second. This means that the data base or file that now takes an hour to transmit at typical, high-speed 50 kilobit service will be transmitted in one minute. Or a one-billion-byte file (8 billion bits) which now has to be distributed by truck could be transmitted to multiple locations in half an hour. Distributed processing to meet operational requirements will be facilitated by these high speeds, with the side benefits of load leveling among data processing facilities, redundancy of files for security in case of fire or accidents, and redundancy of processing capacity in case of machine outages.

The high data rate networks will provide direct communication links between computers, with no distance restrictions or penalties. As a result, a large data base can be managed at a central computing facility, retaining the simplifications associated with a single control point, but with the capability to distribute parts, or all, of the data base to remote locations. This capability will add new dimensions to the management of data base systems and yield some improved management tools. These data networks will enhance real-time central management of raw materials, purchases, inventories, production schedules, distribution, cost and pricing analyses, credit checks and postings, customer orders, competitor activities, etc. Many of these considerations are just as critical from an international perspective as from a domestic one.

Facsimile communications

SBS market investigations have shown that the availability of high-speed facsimile transmission capability can be expected to open up entirely new opportunities in document distribution from hard copy or memory as well as correspondence and first-class mail alternatives. To meet this requirement, the SBS offering will include the capability for digital facsimile transmission through a customer's SBS network at speeds two to 20 times faster than through current systems.

Teleconferencing

A very important finding in the case studies and other efforts undertaken by SBS is the desire expressed by all levels of management for a teleconferencing capability. The expense, inconvenience, and time associated with frequent travel offer an excellent reason for holding many types of meetings using satellite conferencing, and these factors become even more significant when international travel is addressed. SBS' offering of variable bandwidth on demand will accelerate teleconferencing applications as a cost-effective alternative to a large portion of business travel, and the same capacity that permits high-speed data rates and large voice traffic volume will be available for teleconferencing among multiple locations at very little incremental cost. Teleconferencing applications will range from slow-scan to full-motion color, with the customer deciding which he wishes to use.

INTERNATIONAL ENVIRONMENT

The same user needs that SBS perceives for domestic networks would seem to apply equally well for international applications. Certainly different service costs would be involved and the intensity of usage may not be as great as that for a domestic network, but the advantages to business operations of improved management and control through improved communications would be just as important. The benefit of being able to load level computer operations and take advantage of differences in prime shift hours is equally in evidence whether the computer centers are on the east and west coasts of the United States or in the United States and a European location.

This audience is particularly familiar with the shift within the United States and other countries toward an information-based society, and the ability to communicate or access that information effectively is becoming increasingly important. Many things are now under way or planned which are likely to spur increased improvements in international data communications. The efforts of COMSAT, the international carriers and foreign administrations in offering digital and

packet-switched services are a good example. Further, the plans of Intelsat and the European Space Agency to implement 12/14 GHz satellite systems in the 1980's, most likely with digital transmission, promise a future common carrier environment more conducive to the types of computer communications of interest to this group.

It may be helpful to discuss for a moment the existing industry structure. AT&T has the monopoly for dial-up voice and data services and represents about 60 percent of this $1 billion market. The International Record Carriers (IRC's)—RCA Global communications, ITT World Communications, Western Union International (not affiliated with Western Union), and TRT Telecommunications, account for the remainder. The IRC's provide dial-up and leased telegram/telex message services and some private line voice and data. Each of these carriers deals individually with the appropriate foreign communications agency for the terminating portion of the circuit, and either underseas cable or satellite (provided by COMSAT as a carrier's carrier wholesaler) facilities are used for transmission. The Federal Communications Commission has designated five "gateway cities"—New York, Washington, New Orleans, Miami, and San Francisco, and all international communications must be routed through these locations.

Presently, a customer with intermittent international communications requirements will automatically go by AT&T when he makes a dial-up voice or data call or will use Western Union for the domestic part of his record message, with the international segment apportioned by a pre-determined settlements formula to one of the IRC's. For leased private line voice and data circuits, the customer will deal directly with one of the IRC's (or in some cases, AT&T). The IRC's may "accept" the customer's communications anywhere in the U.S. although they are prohibited from providing services in the normal sense outside the gateway cities. The IRC will then lease the U.S. portion of the circuit from one of the domestic carriers and route it through the nearest gateway offices to the appropriate cablehead or satellite earth station for transmission overseas. In addition to the domestic and international circuit charges, the customer pays for the "backhaul" of the service from the gateway city to the nearest cable head or earth station, e.g., a circuit from Chicago to London might be routed to Washington and then backhauled to the Etam, West Virginia earth station for subsequent transmission.

SBS has generated a great deal of interest in the user community, and questions are frequently asked about how the company plans to interconnect its services internationally. Because we are licensed by the FCC as a domestic carrier, we are restricted from providing service to foreign locations and must rely on the appropriate carriers serving those points. Additionally, SBS has chosen initially to provide services to the 48 continental United States and will not serve Alaska, Hawaii, or Puerto Rico. Nor may we interconnect directly with COMSAT for service because, under present regulations, COMSAT is a "carrier's carrier" and, as such, may deal only with licensed international carriers.

We will, however, keep the customer's needs uppermost in our planning and work cooperatively with international carriers to achieve simpler and more direct connections for the customer's higher speed international data communications. The type of interconnect between an SBS customer's domestic network services and his international locations—including Canada and Mexico—will naturally depend on the type (voice, data) and nature (speed, volume) of the communications and on an evaluation of costs compared with other alternatives. However, under present regulations, the customer may interconnect with the international carrier at his SBS earth station closest to that carrier's gateway office, or he may use facilities totally outside the SBS system, just as he does today, if that is appropriate.

There are some obvious limitations in this approach. Now let me describe briefly the type of service environment SBS would like to see for interconnecting its customers' networks for international services. It will be over two years before SBS is operational and the international environment may be quite different then. While there is no circuit-switched higher speed digital service available, the need for such a service would seem to be clear. Many of the future applications identified by SBS require periodic, bulk data movement, and this holds true for interconnecting with international locations.

The FCC is presently investigating the public benefits in expanding the number of gateway cities in which international carriers may operate and, should a decision to expand result, it may prove to be best technically and economically for SBS to locate earth stations, conceivably its own or those leased to customers, in close proximity to the international earth stations and interconnect directly with the international carriers. This would avoid the expense and lower speed implications associated with backhaul and would permit a much simpler and higher quality international transmission.

The double-hop situation which will be encountered using a combination of domestic and international satellites may pose an initial problem for some users, but our conversations with potential customers indicate that this is far from insurmountable. Indeed, several have either adapted, or are planning, advanced data link protocols which minimize the effect of transmission delay and take advantage of the effective transmission capacity of satellites.

Additionally, SBS would welcome the implementation by foreign carriers of a customer premises earth station service concept in their own countries, along with a wider availability of high quality local distribution facilities. From our perspective, these are needed before uniform high-speed digital international networks are achieved.

SBS would hope that foreign regulatory bodies would prove to be just as responsive to clearly demonstrated customer demands as has the FCC and foster an environment that enables a customer to assemble a uniform communications network capability where there is little operational difference between his domestic and foreign locations.

SBS's efforts to date have focused on the domestic private-line communications requirements of a very specific

customer set—large, geographically-dispersed companies—and we believe that the unique service approach described earlier provides these organizations an innovative network capability for meeting forthcoming communications needs. It is SBS's hope that implementation of this service within the United States will stimulate the international community with whom our customers must interconnect to move to provide compatible facilities and services. And SBS will continue to study the future requirements of the marketplace and make its findings known publicly. It is only through this open exchange of information that the potential international user will benefit.

Emerging markets for satellite data communications in the public service*

by JAMES G. POTTER
Public Service Satellite Consortium
San Diego, California

INTRODUCTION

The Public Service Satellite Consortium (PSSC) recently completed a study for NASA's Goddard Space Flight Center which addresses the basic requirements of four potential telecommunications markets in the public service: the U.S. health care system, elementary and secondary education, American libraries, and that sector of the public service which is concerned with the provision of continuing education to health professionals.[1] The composite requirements of these and several other promising market groups are reviewed in this paper.

The potential demand in 1982 for satellite communication capacity from public service users is projected under the optimistic assumption that an appropriate satellite communication network will be available, a network in which numerous, inexpensive earth stations are located at the point of use. It is also assumed that there will be no "gap" in the development of public service satellite communications, which may occur if NASA's endangered species of experimental satellites, such as ATS-6 and CTS, cease to function adequately before a growing and influential community of public service users graduate to operational service on common carrier facilities.

PSSC's surveys among its members have reconfirmed what other studies have shown: that video applications are likely to predominate.[2] Seven of the ten transponders projected by PSSC to support 1982 requirements are related to video applications. Four of these seven transponders already are committed to support public broadcasting. Six represent new markets.

What is surprising about PSSC's projections is evidence of a growing market for satellite data communications service. While projected 1982 channel capacity is modest (on the order of 90 MHz of bandwidth or three conventional satellite transponders), the potential transmission revenue is substantial. This potential revenue, which is estimated to be on the order of $100 million annually, dwarfs that which is likely to accrue from provision of private-line video service.

* This work was sponsored in part by the Goddard Space Flight Center, Contract NAS 5-23865, under the direction of Dr. Edward A. Wolff.

THE UTILITY OF COMMUNICATION SATELLITE NETWORKS

Throughout the public service there are three recurring needs: improved access, maintenance of quality, and containment of costs. The appropriate application of communications satellite networks could ameliorate each of these concerns. Indeed, low cost communications is a prerequisite for organizational arrangements which depend upon the substitution of communications for transportation to achieve higher productivity.

Synchronous communications satellites have generic properties which make them attractive vehicles for performing certain functions of interest to public service organizations:

1. Broadcasting, in which expensive audio-visual programs are aggregated at a few points of origination and distributed to many receivers, as in a TV or radio network.
2. Archiving, in which expensive computer capacity, data bases, audio-visual, and/or computer programs are concentrated at a few central locations where they may be accessed (and updated if appropriate) from a large number of remote points. An example is the computer utility of the Ohio College Library Center in Columbus, which is used by over 1100 libraries in 44 states to construct catalog cards and to purchase serials.
3. Flexible routing. It is possible to allocate the available capacity among the possible routes in the network in an extremely flexible manner. Although the demand for private-line communications satellite circuits from public service organizations may be in excess of $100 million in 1982, the geographic distribution of this traffic, its time distribution, the connectivity patterns, and the magnitude of the peak loads are unknown. The communications satellite will provide a cost-effective, "toe in the water" capable of aggregating related private-line requirements throughout a geographic region.

THE PUBLIC SERVICE SATELLITE CONSORTIUM (PSSC)

PSSC is a growing organization of over ninety non-profit public service agencies from the fields of education, health

care, library service, public broadcasting, and related interests. PSSC was created in 1975 to inform the public service user community about the capabilities of satellite technology and to facilitate aggregation of public service telecommunications requirements and resources.

Most individual public service organizations cannot now afford access to the sophisticated information systems presently (or soon to become) available. Before cost-effective information networks which are responsive to individual user requirements can become operational, it will be necessary to aggregate sufficient numbers of users to allow effective negotiations for core service requirements.

PSSC's role is that of a consultant and broker. It is important that the established carriers have access to comprehensive, objective information regarding the communications requirements of PSSC's members and that PSSC's members are aware of important capabilities which the established carriers can—and cannot—provide. Thus far PSSC has worked exclusively with NASA experimental satellites, but within the next year it will begin buying appropriate communications capacity in bulk from established carriers and re-selling this capacity in smaller increments to its members on a non-profit basis.

PSSC has been working with the State of California and an ad hoc committee of federal agencies, which includes the White House Office of Science and Technology Policy, the Agency for International Development, the Appalachian Regional Commission, the Department of Interior, the Department of Health, Education and Welfare, NASA, and the Veterans Administration, to develop an effective program for accelerating the transfer of communications satellite technology to the public service in the 1980s. Two alternatives are under discussion: (1) use of federal funds to place an appropriate communications payload on the Syncom IV satellite of the Hughes Aircraft Company, which will be launched by the Space Shuttle in 1980; and (2) a competitive service procurement involving established carriers and perhaps aerospace companies, whereby the federal government will provide incentives for the private sector to meet projected public service network requirements using relatively inexpensive satellite earth stations. There is agreement that the growing momentum within the public service community to use cost-saving communications technology must be nurtured without interruption and that public service users must pay some, and eventually all, of the fair price of providing needed service through established common carrier mechanisms.

BASIS FOR DEMAND PROJECTIONS

The projections to follow, while speculative, are based on two assumptions: that significant increases in telecommunications utilization will not occur over a five-year period unless:

1. The organization obviously will benefit from changing its way of doing business, and
2. The required organizational tremors will be mild.

The public service sector has been partitioned into three categories:

1. Category A—modest institutional adjustments are necessary and significant productivity gains are likely, resulting in "modest" risk to the supplier of telecommunications services. (More specifically, the risk is probably still too high for a common carrier in the absence of federal participation to reduce his downside risk; but the likelihood of successful market aggregation is relatively high.)
2. Category B—the institutional requirements picture shows promise, but more information is needed to assess probable benefits and risk.
3. Category C—major institutional adjustments are necessary, the risks to the communications supplier are high, but the possible benefits are also high. Examples include an electronic mail system tailored to the requirements of the U.S. Postal Service, or a telecommunications/computer system addressed to core curriculum subjects in elementary and secondary education, freeing the classroom teacher to concentrate on individual problems rather than mass problems. Small scale experiments which were addressed to "Category C" problems would be appropriate for inclusion in a NASA experimental program, although it may be years before a viable commercial market develops.

The time frame for the projections is 1982, using the best 1976 data available to PSSC. In general, it is assumed that approximately one-third of the total projected telecommunications traffic would be carried by a communications satellite if an appropriate network were available.

The projected levels of telecommunications utilization are based on estimates of what it costs the organization to do a job now—inefficiently. PSSC assumes, for example, that hospitals should not have to spend 10 percent of their annual operating budget on record-keeping functions and is attempting to promote increased use of information networks on this basis. PSSC is assuming implicitly, however, that once a hospital commits to increased use of computers and telecommunications, it will begin using the extra capabilities thereby made available to perform new, unforeseen tasks.

U.S. HEALTH-CARE SYSTEM

PSSC concluded in its study for NASA that the three opportunities which are most likely to lead to extensive utilization of satellite communications in the U.S. health-care system involve the aggregation and sharing of:

1. Computer power, data bases, and software for use in hospital information systems for accounting, billing, inventory control, and patient records;
2. Clinical-support systems for radiology, cardiology, and pathology; and
3. Audio-visual materials and programs for computer-managed instruction to support the continuing educa-

tion requirements of various professional groups who work in a hospital environment.

The U.S. health-care system has been plagued by rampant inflation. Whereas the Consumer Price Index increased by 98 percent in the period from 1950 to 1973, hospital costs increased by 600 percent.[3] In this same period, the federal contribution to health-care costs increased from 25 to 40 percent.[4] In an effort to contain these costs, the Carter Administration attempted to place a ceiling on the rate of increase in hospital expenditures of 9 percent a year in 1977. Although this proposal was not approved by Congress, the health-care industry was placed on notice that effective, voluntary cost-containment procedures must be developed or mandatory federal regulations are likely.

The average American hospital is estimated to spend $20 per patient per day on record keeping,[5] an expense which amounted to $5.6 billion in 1976, or 10 percent of total expenditures.[6] A number of companies, including Technicon, Shared Medical Services, Computer Sciences Corporation, and McDonnell-Douglas Automation, have developed effective automated record keeping procedures which have resulted in productivity gains in participating hospitals.[7] Unfortunately, present indications are that only hospitals having in excess of 200 beds can justify the expense of a dedicated management-information system or can afford access to a shared system. Yet, 70 percent of the 7,174 hospitals in the U.S., many of which are located in rural areas, have less than 200 beds.[8] The availability of an appropriate satellite data communications network would permit a larger fraction of U.S. hospitals to access the automated administrative-support systems which are now being used successfully by larger, urban hospitals.

The advantages of an automated hospital information system include: (1) reduced clerical personnel costs; (2) reduced incidence of lost charges and rejected claims; and (3) improved cash flow. Since such a large portion of health-care expenditures are reimbursed by a third party (e.g., Medicare, Blue Cross, or Blue Shield), use of central data bases and electronic information transfer can lead to a dramatic improvement in cash flow. The average payment cycle is reported to have been reduced from sixty days to twelve days in several California hospitals which recently implemented automated record keeping systems.[9] Other advantages of hospital information systems which include the patient history are: (1) improved professional communications, which enhances continuity of care (the possibility of lost or incomplete records is reduced) and reduces the incidence of episodic care; (2) groups of patients at risk can be identified, and preventive medicine can be practiced systematically; (3) the quality of care can be audited more easily; and (4) the continuing education needs of the providers can be determined more systematically.

The disadvantages of automated record keeping systems are: (1) there are potential privacy problems (although no system is immune from unauthorized access); (2) the relative strengths and weaknesses of providers can be evaluated more systematically, which may or may not create barriers to acceptance; and (3) such systems require structured input, which necessitates annoying changes in the working routine of health practitioners.

In projecting the magnitude of the satellite data communications market in 1982, PSSC makes the following assumptions:

1. The rate of growth of hospital expenditures will stabilize at a compounded annual rate of 9 percent, as requested by the Carter Administration. (The rate of growth in 1975 and 1976 was about 14 percent, and total hospital expenditures amounted to $55.4 billion in 1976.[10])
2. Economies of scale resulting from use of a satellite data communications network will make hospital information systems cost effective for hospitals having at least fifty beds. (Hospitals having more than fifty beds accounted for 97 percent of total expenditures in 1976.[11]).
3. Hospitals will continue to spend approximately 10 percent of their total budget on record keeping functions.
4. The percentage of the record keeping budget spent on automation (given that automated procedures are used at all) will remain at its present value of 12 percent.[12]
5. The percentage of the automated record keeping budget spent on telecommunications will be 15 percent.[13]
6. The percentage of telecommunications service provided by satellite will be 33 percent.

The projected 1982 satellite revenue is then: $(\$55.4B)(1.09)^5(0.97)(0.12)(0.15)(0.33)=\$49$ million.

Preliminary information available to PSSC suggests that an American hospital performs an average of sixty-two record keeping transactions per patient per day.[14] Approximately 270 million patient-days of service were administered by American hospitals having fifty beds or more in 1976. Assuming that the volume of traffic grows at a compounded rate of 9 percent, that 50 percent of the total volume is processed locally, that 33 percent of the remaining volume is carried by satellite, and that the average administrative form contains 8,000 bits but that only 50 percent of this information content needs to be transferred back to the data base in an average transaction, the estimated volume of hospital administrative traffic in 1982 is: $(270\times10^6)(1.09)^5(62)(0.5)(0.33)(8{,}000)(0.5)=1.7\times10^{13}$ bits per year.

PSSC has not included possible traffic associated with clinical support services in its estimates. Three areas which show near-term promise are radiology, cardiology, and pathology. PSSC is evaluating these areas but does not now have sufficient information to estimate probable demand.

One must exercise caution when projecting demand for telecommunications in clinical medicine. It would be logical to consolidate health-care delivery throughout a geographic region, substituting communications for travel and treating patients at the lowest possible level of a hierarchical health-care system in the interest of convenience, fairness, and cost. Dr. Maxine Rockoff of the National Center for Health Services Research notes a basic fallacy in this line of think-

ing, however:[15]

"To the extent that the cost of care increases with each level in the hierarchy, avoiding referrals reduces costs, and benefits those who pay for care, including patients and insurers. But considered from the perspective of that 'next level up' whose expertise is to be brought to the patient via telecommunications technology instead of having the patient referred to it, this may be no benefit at all. Indeed, the pecuniary interests of the 'next level up' may be best served by maximizing referrals, not minimizing them."

A possible implication for those who wish to market telecommunications services to clinicians is that one should concentrate on the sharing of computer and audio-visual resources, not scarce human resources.

CONTINUING EDUCATION

In a survey which was administered in the summer of 1976, 81 percent of PSSC's membership expressed interest in a network for continuing education. More recently, PSSC conducted a survey of the membership to determine composite requirements for a federally subsidized Public Service Communications Technology Satellite (presumably Syncom IV with an appropriate communications payload). In the period between October 17, 1977 and January 10, 1978, forty-two public service organizations responded in writing or were interviewed by senior staff. Quantitative information regarding the probable demand for private-line video, voice, and data circuits was received from seventeen organizations.

The response was strongly oriented toward one-way video. If service which is comparable to that planned for Syncom IV (which is expected to provide an EIRP in excess of 43 dBW throughout the continental U.S.) is available at approximately $400 per transponder-hour, an estimated 8,170 channel-hours would be utilized annually by these seventeen organizations. On the basis of this response, PSSC projects a demand for 12,000 channel-hours of one-way video in 1982, which represents an annual revenue of approximately $4,800,000. Although the time distribution and peak load of this requirement is unknown, PSSC estimates that three protected transponders will provide adequate service.

PSSC plans to respond to this requirement in 1979 by using facilities of the Western Union and RCA satellite networks. Westar will interconnect one hundred and fifty Public Broadcasting Service earth stations and one hundred and ninety-two National Public Radio stations.[16] PSSC is now engaged in a study for the Corporation for Public Broadcasting to develop equitable arrangements for sharing the public broadcasting network. PSSC plans to supplement coverage available from Westar using RCA's Satcom network, which now interconnects approximately one hundred and eighty cable TV systems.

When economical service becomes available, PSSC plans to develop a continuing education network which provides one-way television and two-way data to each classroom. Much of the material to be distributed will be pre-recorded, but an important element of the programs will be interaction involving use of mark-sense cards and optical card readers. Prior to the class, the students will receive study guides which contain a number of questions. The answers to these questions and others raised during the class will be answered in a multiple-choice format and relayed to the studio during the class. The lecturer and his assistant will receive a histogram of responses in real-time, which will help to determine what supplementary material to emphasize. Each student's responses will be evaluated in non-real time by computer, and each student will be mailed an evaluation after the course is completed. Although this evaluation will be computer-generated, resources in the student's community will be identified to follow up progress made during the class.

The projected volume of data communications traffic is modest. Assuming that 12,000 channel-hours of programming are delivered annually, that there are an average of 1,000 students per class, that the average student responds 16 times an hour, and that 30 bits of data are transferred per response, only 6×10^9 bits per year of data need be transferred while maintaining individual records. The required system throughput is so low that the economics are significantly enhanced by relying exclusively on mark-sense cards and not allowing voice feedback, which tends to be ineffective in large-audience situations anyway.

A one-way video and two-way data network could combine the best features of pre-recorded lectures, live interaction, and computer-managed instruction. Even with production costs as high as $20,000 per hour and transponder costs of $400 per hour, it would be possible to administer such a network at a profit while charging course fees of approximately six dollars per student-hour if an average of 1,000 students per class could be attracted.[17]

EQUIPMENT MAINTENANCE

Federal and state government agencies have extensive investments in equipment which is dispersed throughout the country. PSSC does not have adequate information regarding the dimensions of this requirement, but preliminary insight may be obtained from the experiences of the Federal Aviation Administration in its VORTAC (VHF Omni Range Tactical Communications) system.[18] In 1975 the FAA had a total maintenance budget of $320 million, of which 80 percent was personnel related. Through use of telemetry and centralization in its VORTAC program, the FAA could eliminate the need for many on-site technicians and spare parts, which would result in a considerable savings. Studies by the Mitre Corporation indicate that the maintenance budget for the VORTAC program, which amounted to $38 million in 1975, could be reduced to approximately $10 million annually through appropriate use of telecommunications.[19]

Use of telemetry, microprocessors, and telecommunications probably could effect tremendous savings in equipment

maintenance and load management of various resources (power, water, light, heat, air conditioning, oil and gas pipeline flow, control of traffic lights and vehicular flow, etc.). To gain a tentative estimate of the volume of the associated telecommunications service, PSSC makes the following assumptions:

1. The total investment in capital equipment by public service agencies is $100 billion, an amount which is growing at 5 percent annually;
2. Five percent of this equipment is subject to savings through appropriate use of telemetry and centralized maintenance procedures;
3. The annual cost of maintenance averages out to be 10 percent of the capital cost of the equipment;
4. In those cases where savings can be effected through use of telemetry, 15 percent of the maintenance budget is associated with telecommunications service; and
5. One-third of the resulting traffic is carried by satellite.

The resulting projected potential satellite traffic is: $(\$100B)(1.05)^5(0.05)(0.1)(0.15)(0.33)=\$32$ million.

The annual bit rate is projected under the assumption that the average cost of this dispersed equipment is $25,000, that thirty-two bits of data are sufficient to address and encode the message, that each piece of equipment is interrogated every ten minutes on the average, and that all of this information is relayed by satellite: $(\$100B/\$25K)(1.05)^5(0.05)(32)(6)(24)(364)=4.3\times10^{11}$ Bits/Year.

AMERICAN LIBRARIES

Requirements to support so-called "technical services" in American libraries (cataloging, serials control, acquisitions, circulation, and inter-library loans) will be projected from the present budget of the Ohio College Library Center (OCLC), which now spends approximately $2,280,000 annually on leased terrestrial lines to reach 1,100 libraries in 44 states. PSSC makes the following assumptions:

1. Use of automated services in libraries which already accept the concept will grow from a present level of 3 percent of the total operating budget to 8 percent of the operating budget.[20]
2. The number of libraries which use automation will grow at a compounded annual rate of 15 percent; and
3. One-third of the total telecommunications traffic potentially could be routed by satellite.

The potential revenue in 1982 for technical services is estimated to be $(\$2,280,000)(1.15)^5(8.0/3.0)(0.33)=\$4$ million.

5,707,828 books were cataloged in the OCLC system in 1976, using an average of 4,800 bits per request.[21] PSSC projects the potential volume of satellite traffic in 1982 to be $(5,707,828)(1.15)^5(8.0/3.0)(0.33)(4,800)=4.8\times10^{10}$ Bits/Year.

To gain a qualitative appreciation of the potential demand for information retrieval services in 1982, PSSC makes the following assumptions:

1. One percent of the U.S. population will interrogate a data base on the average of once a week;
2. The average cost of each search will be $2;
3. Fifteen percent of this revenue will be related to telecommunications; and
4. Thirty-three percent of the traffic will be routed by satellite.

The estimated revenue is then $(220\times10^6)(\$2)(52)(0.01)(0.15)(0.33)=\$11$M.

Assuming that an average of ten bibliographic records are retrieved with each search, each of which is encoded with 4,800 bits, the potential volume of satellite traffic is $(220\times10^6)(52)(0.01)(0.33)(10)(4,800)=1.0\times10^{12}$ Bits/Year.

The total revenue deriving from library-related services is estimated to be $15 million.

ENVIRONMENTAL MONITORING

The National Oceanic and Atmospheric Administration (principally the National Weather Service), the Environmental Protection Agency, the U.S. Geological Survey, the Corps of Engineers, the Department of Agriculture, and the Department of Interior all have extensive requirements for monitoring atmospheric, edaphic, and/or oceanic data. Ecosystems International Inc., under contract to the Goddard Space Flight Center, determined that in 1975 these agencies maintained 90,714 stations in the U.S. having an average of four sensors apiece.[22] The annual cost of the present operation is estimated to be $98.3 million and the volume of information is 1.36×10^{11} Bits/Year. Only 6 percent of these platforms are remotely interrogated at the present time. PSSC estimates that if an appropriate satellite data communication network were available, environmental monitoring would contribute another $5 million of business annually. In arriving at this estimate, it is assumed that 10 percent of the present $100 million budget would be spent on telecommunications services, of which half would be directed to satellite carriers.

OTHER POTENTIAL MARKETS

PSSC does not have sufficient information to evaluate a number of other promising candidates for service. Other possibilities are:

- Department of Defense training
- Training of other federal and state employees
- Law enforcement (fingerprint data, automobile registration and license information, LETS, and NCIC functions)
- Access to data regarding eligibility for welfare benefits
- Search and rescue/disaster relief
- Internal Revenue Service, Social Security data transfer
- Job bank involving the Department of Labor

TABLE I.—Projected Market for Public Service Satellite Communications (1982)

Service Sector	Est. Revenue	Est. Volume
American Hospitals	$ 49M	1.7×10^{13} Bits/Year
Equipment Maintenance	$ 32M	4.3×10^{11} Bits/Year
American Libraries	$ 15M	1.0×10^{12} Bits/Year
Continuing Education	$ 5M	Three video channels 5.8×10^{9} Bits/Year data
Public Broadcasting	$ 4M	Four video channels Four 15 kHz audio channels
Environmental Monitoring	$ 5M	6.8×10^{10} Bits/Year
Total:	$110M	Four Western Union transponders Six additional transponders

SUMMARY OF TRAFFIC PROJECTIONS

PSSC's projections of "Category A" service sectors are summarized in Table I. Seven video transponders, which represent space-segment revenues of $9 million annually, and three additional transponders for data, which represent total transmission revenues of $101 million, are projected. Four of the projected seven video transponders will be used by public broadcasting, which has entered into a seven-year contract with Western Union. In addition, approximately one-third of a Western Union transponder will be used by National Public Radio to distribute four 15 kHz audio channels to its licensees.

In a large network of small stations, the space-segment revenue associated with the estimated data traffic will amount to approximately 20-30 percent of the total, or $20-$30 million. It is noteworthy that such a small amount of bandwidth could generate so large a revenue—given adequate market aggregation.

COMMENTS ON THE NATURE OF THE TRAFFIC

PSSC's near-term market projections are dominated by the requirements of the health-care industry, both in terms of volume and revenue. Public education is conspicuous by its absence. Dr. Louis A. Bransford, Director of Service Development of PSSC, concluded in the recent market study for NASA:[23]

"The structure of the present system of public education in America, both economically and programmatically, appears to be inconsistent with the requirements of a broadly-based telecommunications network. Implementation of a comprehensive information network may face organized resistance and probably will take years to accomplish."

The projected volume of satellite data communications traffic is on the order of 2×10^{13} bits per year. To place this figure in perspective, a single Satellite Business Systems transponder operating continuously at 50 megabits per second has a peak capacity of 1.6×10^{15} bits per year, about an order of magnitude higher than that required to support projected public service requirements, assuming that the peak load will be about ten times the average load. The earth stations which SBS allegedly plans to install have a capital cost of approximately $474,000.[24] The possibility of exchanging reduced system throughput for reduced earth-station cost immediately suggests itself.

Table II was suggested by Norman Abramson of the Aloha Systems Project, who is an expert on "affordable" satellite data communications.[25] Dr. Abramson advocates use of a packet-switched satellite data communications network, operating at a speed somewhere between one and two orders of magnitude below the SBS network. For concreteness, Table II assumes a system throughput of five megabits, although this figure is arbitrary and not the result of a cost-performance evaluation.

In a network which is composed of many point-to-point links which interconnect a large number of nodes, such as the Bell System's Direct Distance Dialing Network for message telephone service, it is not economical to connect every node to every other node. Rather, the messages are routed through a hierarchy of nodes, using expensive switching machines to provide the necessary connectivity. When the application calls for direct connection of nodes, as in broadcasting or archiving, the additional links and switching machines of a hierarchical network add unnecessary cost, complexity, and noise to the system.

Dr. Abramson observes:[26]

"The natural structure of satellite communications links does not require the establishment of point-to-point communications channels in the traditional sense. A more natural form for satellite communications resources is a broadcast structure, allowing each network node to communicate directly with every other network node. The communications architecture which best matches the broadcast structure of the satellite communications channel is therefore one which starts from the premise that communications can proceed from many transmitters to many receivers in a communications environment analogous to that of a multi-person conference. And in the case of data communications it is possible to implement such a broadcast architecture using small earth stations."

TABLE II.—An Affordable Satellite Data Communications Network

Application	Capacity	Capacity Using Low-Cost Earth Stations
Continuous Data	50 Mbps	5 Mbps
Two-Way Video	One Conversation	Not Applicable*
Two-Way Audio	800 Conversations	80 Conversations
3,000 Word Reports	400 Reports/Sec.	40 Reports/Sec.
200 Word Messages	6,250 Messages/Sec.	625 Messages/Sec.
Interactive Computer Terminals	500,000 Active Users	50,000 Active Users

* Two-way video would not be physically impossible in this hypothetical network, but the earth stations probably would cost at least $85,000 apiece. PSSC assumes arbitrarily that earth stations must cost less than $25,000 in an "affordable" satellite communications network.

To make a satellite data communications network for modest-throughput requirements more affordable, Dr. Abramson suggests that: (1) multiple access of the transponder be accomplished using Aloha-type packet-switching techniques; (2) the satellite transponder be designed to transmit at a peak power which is approximately ten times higher than the average power; and (3) that such transponders use relatively narrow bandwidths (about two MHz). With these modifications, the required sensitivity of two-way data earth stations could be reduced substantially and/or the system throughput could be maintained at levels which approach those attainable with more expensive TDMA multiple-access procedures. But if conventional satellite transponders are used in the early 1980s to meet public service data communications requirements, it is still likely that the earth stations can be built in lots of 50 for less than $25,000 apiece. Hughes Aircraft already has developed a similar earth station for operation at four and six GHz which it plans to market for about $25,000.[27]

CONCLUSION

The communications capacity required to serve some important public service requirements is modest, and the potential revenue is significant. What is needed is a cooperative effort by common carriers and major public service institutions to aggregate the market. The federal government may or may not elect to accelerate the communications-technology transfer process by providing initial subsidies. In any case, PSSC will continue to focus on the requirements of its members and to impress on the carriers that there is money to be made by responding to these requirements.

ACKNOWLEDGMENTS

PSSC's information regarding the requirements of the health-care industry was assembled primarily by Dr. Thomas E. Terrill of the Akron City Hospital in Akron, Ohio. The material concerning continuing education was synthesized by Ms. Kathleen King of PSSC's staff and by Ralph P. Christenson, M.D. I am indebted to Dr. Norman Abramson of the University of Hawaii and to Gregory Nichols of PSSC's staff for numerous insights concerning network architecture and systems engineering. To John Witherspoon, President of PSSC, I owe most of the insights I have obtained regarding the institutional factors which will be the predominant influence underlying the ultimate acceptability of the emerging communications techniques which now show such promise in the public service.

REFERENCES

1. Public Service Satellite Consortium, *Developing Satellite Communications for Public Service: Prospects in Four Service Areas,* sponsored by Goddard Space Flight Center, Contract NAS 5-23865, Accession Number N78-10347, San Diego, California, September 30, 1977.
2. FCC Broadcast Bureau Service Group on Satellite Broadcasting, "Need Projections and Spectrum Requirements," WARC-79, Washington, D.C., Federal Communications Commission, June 1976.
3. U.S. Department of Commerce, Bureau of the Census, *Statistical Abstract of the U.S. 1974,* Washington, D.C., U.S. Government Printing Office, 1974, tables 98, 665, 666.
4. Worthington, Nancy, "National Health Expenditures, 1929-1974," *Social Security Bulletin,* February, 1975, p. 9-13.
5. Deland, E. O. and B. D. Waxman, "Review of Hospital Information Systems," *Hospital Information Systems* (Edited by George A. Bakey and Morton D. Schwartz). Biomedical Engineering, A Series of Monographs, (New York: Marcel Dekker, Inc., 1972), Vol. 1, pp. 1-38. The estimate of $20 per patient per day for record keeping, which was derived by Deland and Waxman using 1971 data, was revalidated by PSSC's consultant in hospital administration, Dr. Thomas E. Terrill of the Akron City Hospital, in March 1977.
6. Gibson, Robert M. and Margorie Smith Mueller, "National Health Expenditures Fiscal Year 1976," *Social Security Bulletin,* April 1977, U.S. Government Printing Office, p. 1.
7. National Center for Health Services Research, "Demonstration and Evaluation of a Total Hospital Information System," Research Summary Series, DHEW Publication No. (HRA) 77-3188, Hyattsville, Maryland, July 1977, p. 25.
8. American Hospital Association, *Hospital Statistics 1975 Edition,* Chicago, Illinois, 1975, p. 10.
9. Commissioner John R. Burgis, California Health Facilities Commission, private conversation with James G. Potter, December 22, 1977.
10. *Op. Cit.,* Robert M. Gibson and Margorie Smith Mueller, p. 1.
11. *Op. Cit.,* American Hospital Association, p. 11.
12. *Op. Cit.,* Public Service Satellite Consortium, p. III-25. PSSC's consultant, Dr. Thomas E. Terrill, obtained his data from Scott Parker, President, Intermountain Health Care Corporation, Salt Lake City, Utah, in March 1977.
13. PSSC's consultant on information retrieval, Martha E. Williams of the University of Illinois, reported in August 1977: *"A rule of thumb in the information industry is that telecommunications accounts for 10 to 15 percent of the total cost of an information service."*
14. The assumptions of this paragraph are based largely on the experiences of Intermountain Health Care, Inc., an 18-unit health-care complex in Utah.
15. Rockoff, Maxine L., "Telecommunications Technology: Can It Lead to Health Care Delivery Reform?," address delivered before the Royal Society of Canada/NASA Conference on the HERMES Communications Satellite, November 29, 1977, National Center for Health Services Research, Hyattsville, Maryland, pp. 4-5.
16. *Electronic News,* January 16, 1978, p. 17.
17. *Op. Cit.,* Public Service Satellite Consortium, pp. VII-12 to VII-16.
18. Meer, Ahmed, "The Economics of Centralizing VORTAC Maintenance (Draft)," McLean, Virginia, The Mitre Corporation, June 1976.
19. *Ibid,* p. 9.
20. Butler, Brett, "Use of Automated Services," *Bulletin of the American Society for Information Sciences,* Vol. 2, No. 3, September-October 1975, pp. 21-22.
21. Schieber, Phillip, The Ohio College Library Center, Columbus, Ohio. Private conversation with James G. Potter, March 30, 1977.
22. Castruccio, Peter, and Harry L. Loats, "Data Collection Systems Requirements: Correlation Study," Gambrills, Maryland, Ecosystems International, Inc., January 1977.
23. *Op. Cit.,* Public Service Satellite Consortium, p. IV-22.
24. Satellite Business Systems, *Applications of Satellite Business Systems for a Domestic Communications Satellite System, Volume I: General Description of SBS Structure and System Design and Operational Policies.* Before the Federal Communications Commission, Washington, D.C., December 1975, p. I-8-2.
25. Norman Abramson, "Palapa for Data Communications," Internal Memorandum, The Aloha System, University of Hawaii, Honolulu, April 1977, p. 3.
26. *Ibid,* p. 10.
27. Tustison, Galen, Hughes Aircraft Company, El Segundo, California. Private conversation with James G. Potter, July 13, 1977.

Packet switching services for the autodin community

by DONALD J. O'ROURKE
United States Government
Washington, DC

INTRODUCTION

The AUTODIN II Program brings the advantages of packet switching technology to the Defense community. The concepts and advantages demonstrated successfully in the AUTODIN I store and forward message switching technology are not being discarded, but rather supplemented by a parallel operating AUTODIN II in order that the best transmission techniques for the efficient transfer of bulk transaction and interactive type data can be made available to the community of data system users.

This paper reviews the management decision to add packet switching capabilities to the AUTODIN I store and forward message environment; examines the technology to

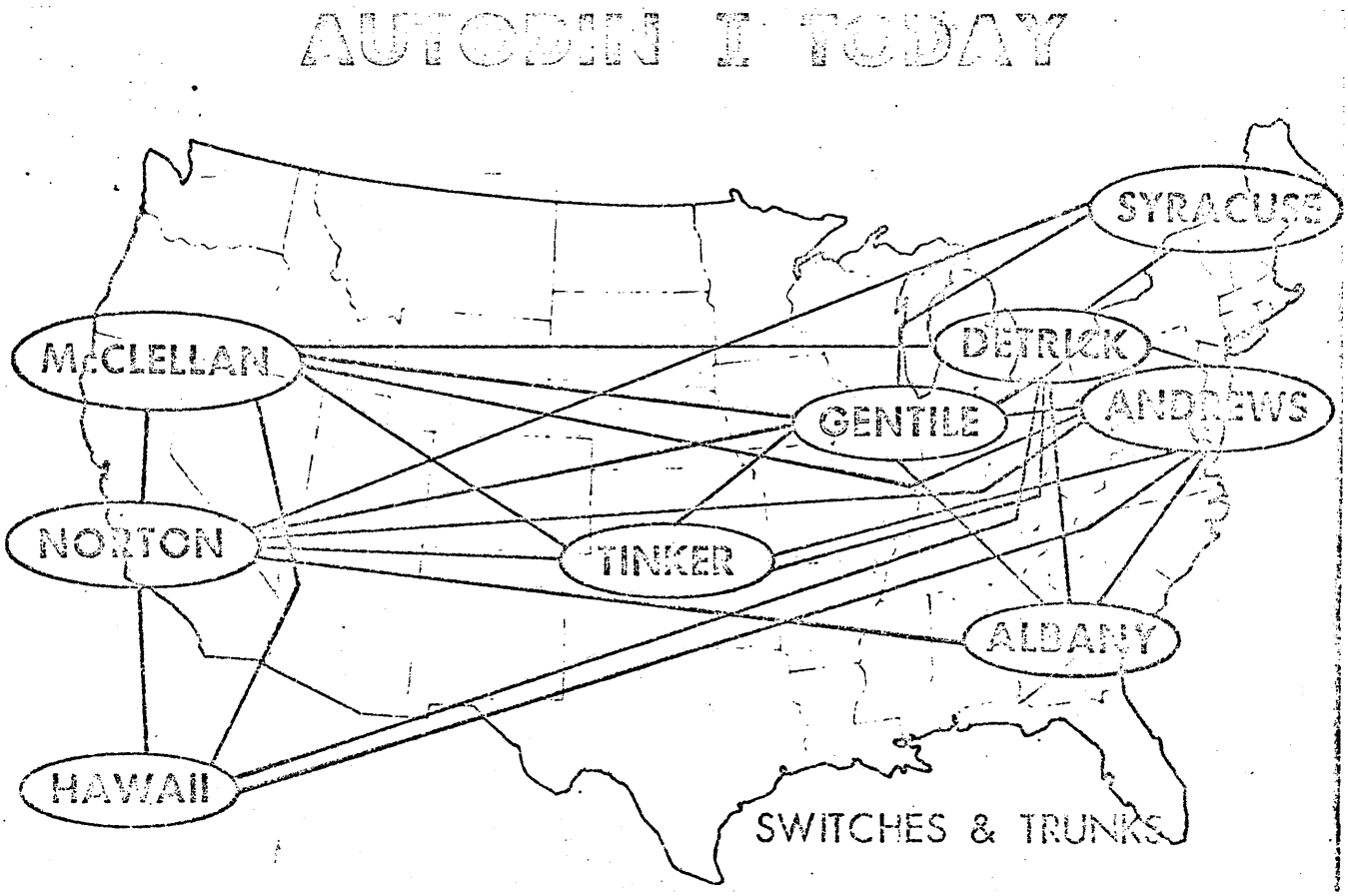


Chart I

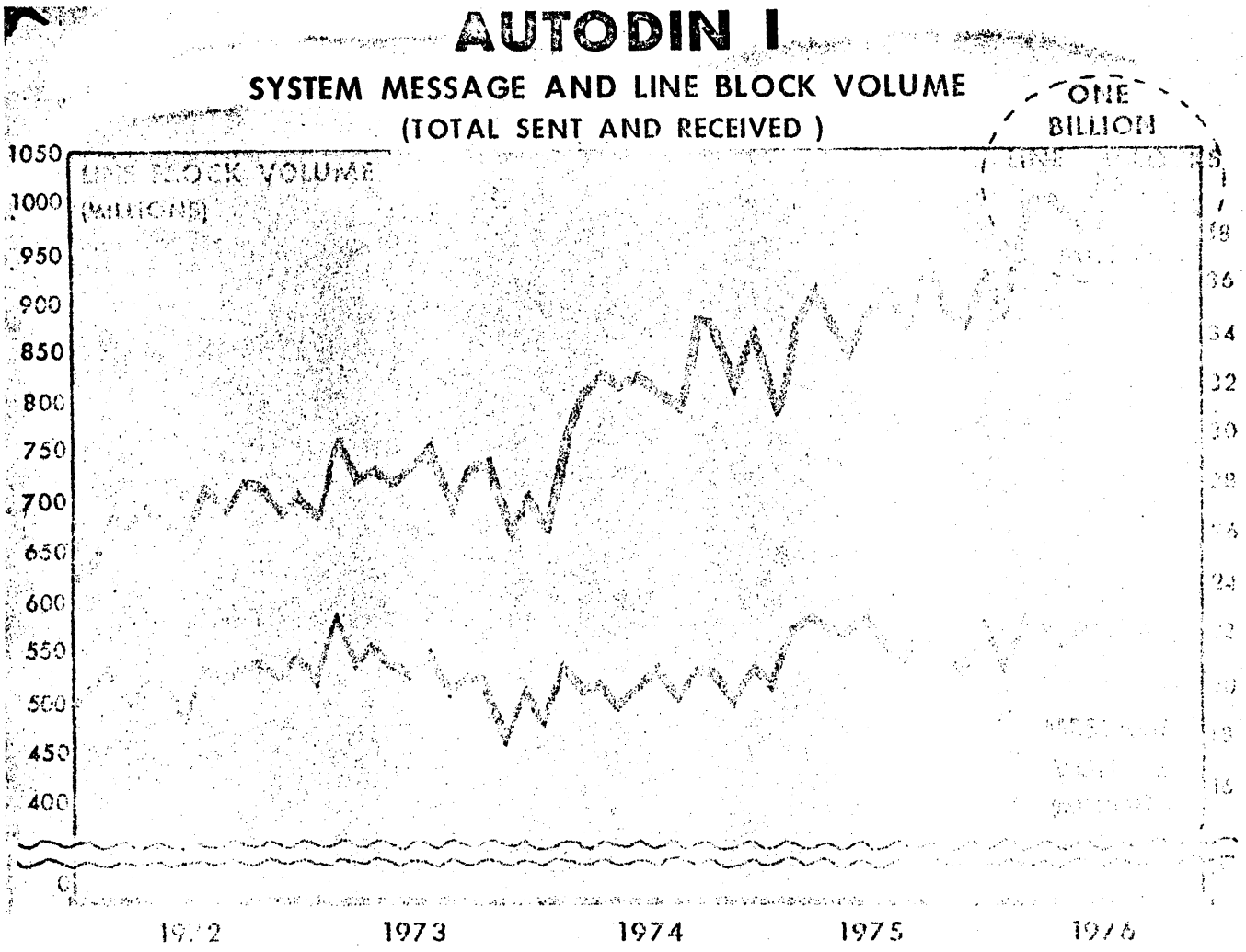


Chart II

be employed, discusses the implications for future growth and utility in this environment, and provides a report on the progress toward realizing that goal.

PLANNING FOR AUTODIN II

Since the advent of data processing, the Federal Government has experienced a steady growth in the numbers of computer based information systems employing remote terminals. The development and implementation of the AUTODIN I in the early 1960s was the first step in providing computer oriented communications network capabilities to this community of users. The AUTODIN I provides message switching store and forward communications on a worldwide basis.

Chart I shows the presently operating switches of the AUTODIN I System within the Continental limits. Eight additional switches located on foreign soil provide the overseas extensions of the AUTODIN I System.

The traffic on the AUTODIN I System has steadily grown through the years, requiring periodic increases in the system's capacity and the expansion in the type of services rendered. While the number of messages shown in Chart II has only grown at an average yearly rate of 5 percent, the type and length of messages have changed drastically over the past five years. The system traffic, measured in lineblocks (each lineblock equals 640 bits), has grown from 625 million lineblocks in 1972 to over one billion lineblocks per month in 1977 (640 billion bits per month). This increase in traffic, plus the characteristics of the traffic itself clearly indicated the increased use of AUTODIN I for computer oriented data transfer.

Studies by the Department of Defense and others have

AUTODIN II

TRAFFIC ANALYSIS

	USAGE AT INSTALLATION		POSSIBLE PROJECTED USAGE IN THE '80'S
	% BUSY HOUR PACKETS	% BUSY HOUR BITS	
O INTERACTIVE	6	1.2	45.0
O QUERY/RESPONSE	1	.5	
O BULK DATA	79	85.1	40.0
O NARRATIVE	5	4.5	10.0
O AUTODIN I TRUNKING	9	8.7	5.0
O OTHER NETWORKS	--	--	

Chart III

shown a continuing trend toward the growth of transaction oriented interactive systems, in addition to the established requirement for remote job entry and bulk data and narrative transmission. A study initiated in 1972 indicated that by 1976 there would be approximately 250 computers involved in on-line communications functions with approximately 8,000 terminals associated with these systems. Projections into the mid-1980s indicated that within the DOD, there will be approximately 2,500 computers with some 20,000 terminals involved. These studies led to an examination of the methodologies to be employed to meet these requirements. Among the earliest conclusions was the decision that to the fullest extent possible common-user rather than dedicated networks must be employed to ensure the most efficient and economical service possible.

- HIGH TRUNK UTILIZATION
- NO BUSY CONDITIONS (System accepts traffic-stores until transmitted.)
- MESSAGE RESPONSIBILITY
- SPEED, CODE AND FORMAT CONVERSION
- MESSAGE RETRIEVAL CAPABILITY
- MULTIPLE ADDRESSING
- INEFFICIENT FOR MIXTURE OF INTERACTIVE AND TRANSACTION ORIENTED OR PRECEDENCE GOVERNMENT TRAFFIC.

Chart IV—Characteristics of Message Switching vs. Circuit Switching

- EXCELLENT FOR INTERACTIVE AND TRANSACTION ORIENTED OR PRECEDENCE GOVERNMENT TRAFFIC
- TRANSPARENT TO ALL USERS DESPITE USER PROTOCOL
- HIGHER SPEED—LOWER ERROR RATE
- HIGH RELIABILITY—ADAPTIVE ROUTING
- RAPID TRANSIT THROUGH SYSTEM
- NO MESSAGE RESPONSIBILITY
- NO MESSAGE RETRIEVAL OR RECOVERY
- HIGHER OVERHEAD REQUIREMENTS
- TRAFFIC HELD AT SUBSCRIBER VIAL FLOW CONTROL
- SMALL INTRANSIT STORAGE

Chart IV A—Characteristics of Packet Switching vs. Message Switching

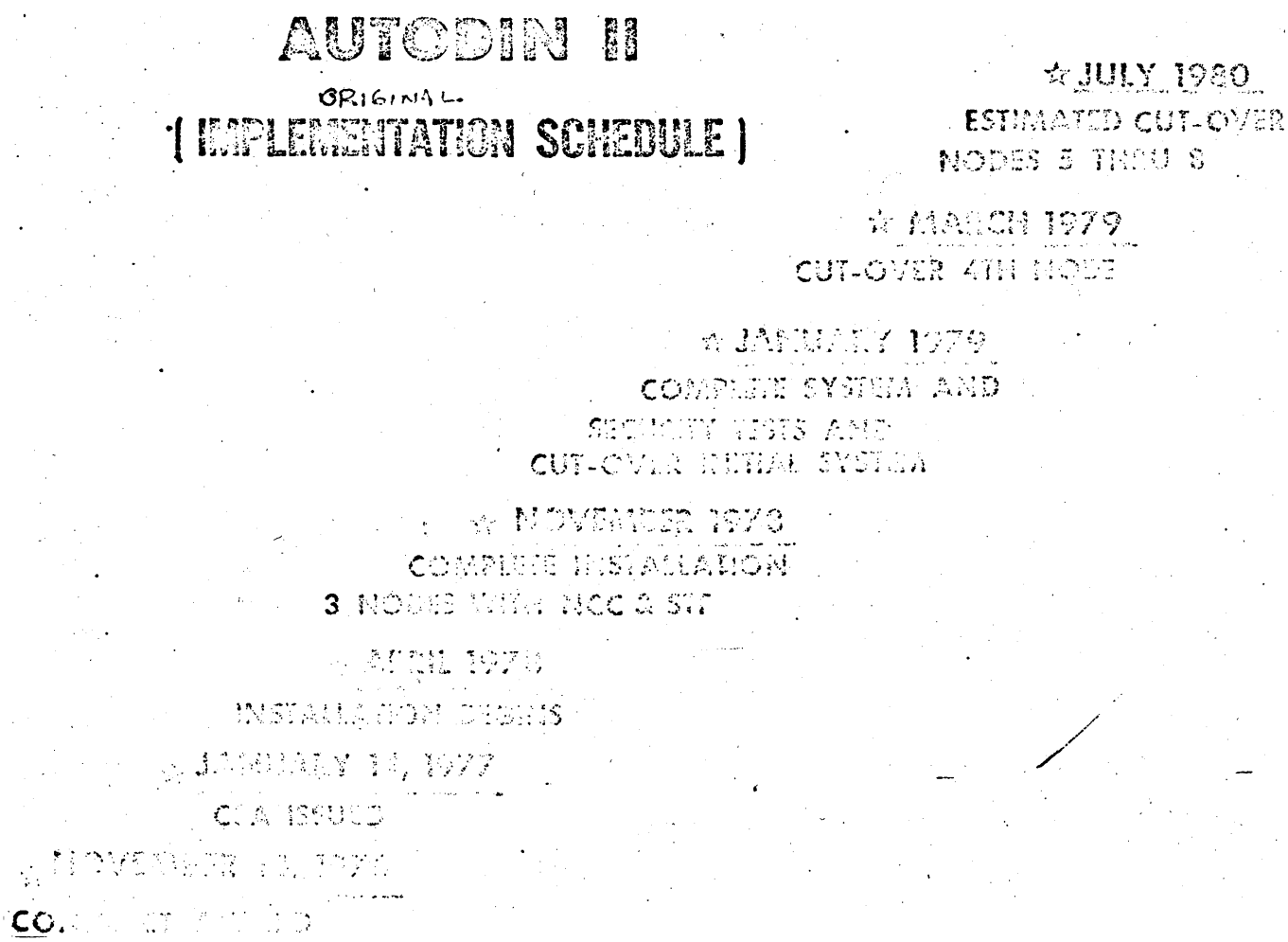


Chart V

The economies of AUTODIN I, as a common-user message system, have been well documented. With the possible demise of TELPAK rates and their replacement by higher cost facilities, the decision by the Defense Department to continue their dedication to the common-user network concept through the implementation of AUTODIN II, is even more commendable in today's world than it was even just five years ago when the basic design concepts of DIN II were originally conceived. A study of the various switching alternatives highlighted the characteristics of Circuit, Message and Pocket Switching.

An examination of these summaries suggests that there is an advantage to maintaining a store and forward capability in the Federal Communications environment for message type traffic while adding packet switching capabilities to satisfy the requirements of the interactive and transaction oriented user.

The trend toward interactive type traffic, as opposed to bulk transfer, has been clearly demonstrated (Chart III). While the initial traffic introduced into the AUTODIN II System shows that 85 percent of it is devoted to the transfer of bulk information, the amount of interactive and transaction oriented requirements will greatly expand so that by the early 1980s, a cross section of the AUTODIN II traffic will show that 45 percent will be in this category of service. Since packet switching, as indicated on Chart IVA, provides the most efficient transmission scheme for this mixture of traffic, the decision by the Department of Defense to implement a packet switching network, is commendable. Accordingly, a decision was reached to acquire packet switching network capabilities and to have that network (AUTODIN II) work in conjunction with AUTODIN I. High speed interconnect channels will connect the two systems together. The message type traffic handled by AUTODIN I will be transferred over these dedicated links to AUTODIN II and transmitted over a common high speed transmission net-

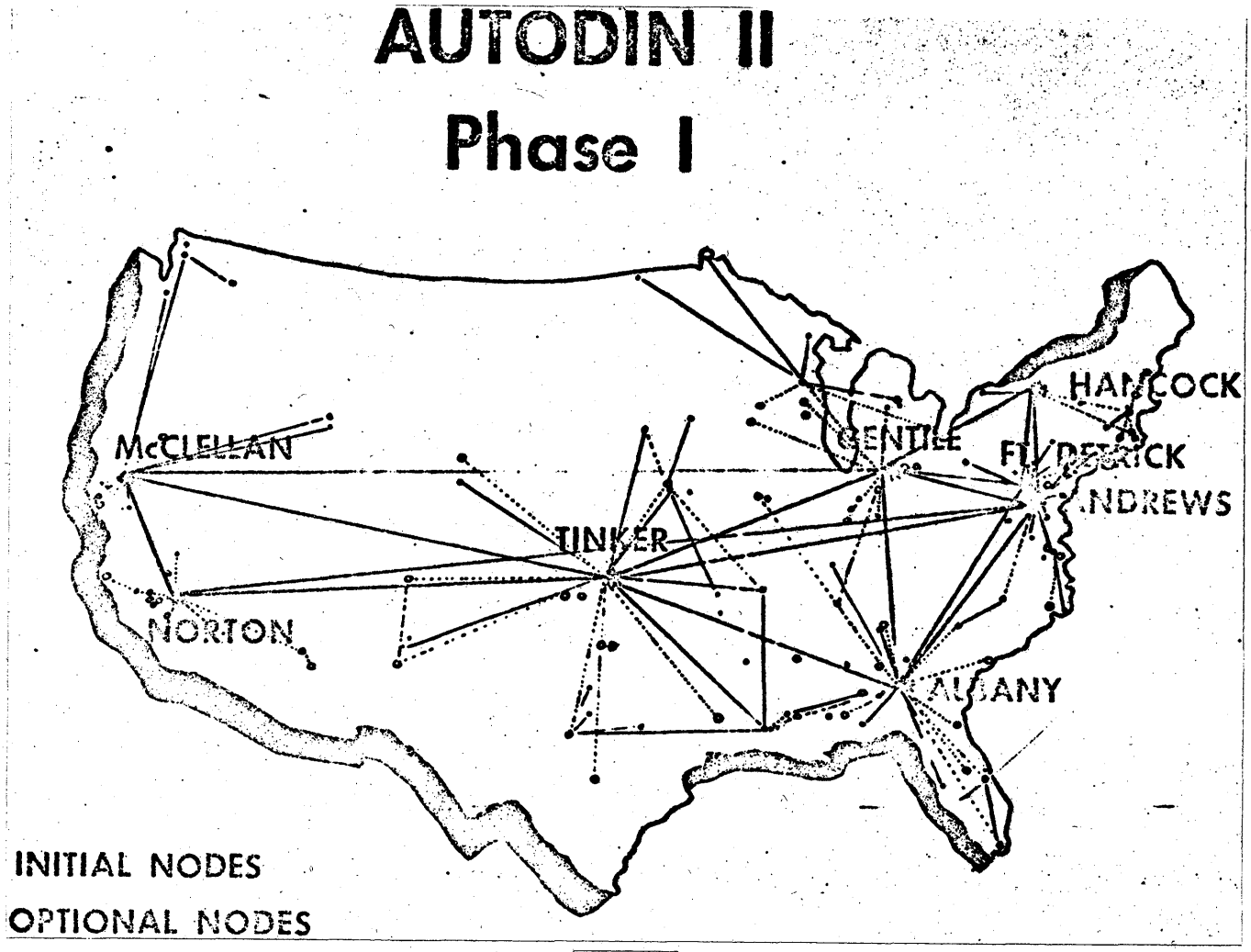


Chart VI

work, thereby providing to the users the choice of two types of data switching facilities through a common transmission network.

The request for proposal for this service was released to industry on November 26, 1975. An award of a contract for development and implementation of the system was made to The Western Union Telegraph Company on November 10, 1976 with the Commercial Service Order issued January 1977. Computer Sciences Corporation and Ford Aerospace and Communications Corporation are the primary sub-contractors to Western Union. (Chart V)

The contract awarded for AUTODIN II, Phase I, covers the CONUS operation only. Phase II will expand AUTODIN II into overseas locations. Until Phase II is implemented, the overseas AUTODIN I System will handle data traffic and will be interconnected into the AUTODIN II System overseas at suitable gateway locations. (Chart VI)

TECHNOLOGY EMPLOYED

At this point it is advisable to ask "What is Packet Switching?" The answer could fill volumes but, here we will restrict ourselves to the CCITT definition (Chart VII).

"A group of binary digits, including data and call control signals (address) which is switched as a composite whole. The data, call control signals and possibly error control information, are arranged in a specified format." Essentially, each addressed packet occupies a transmission channel for the duration of the transmission of the packet only. The channel is then available for use by packets being transferred between other users. Thus, channels between packet switches are shared between many users on a demand basis, and it becomes possible for a user having a single link to engage in the exchange of data packets with a number of other users at the same time. This technique results in the

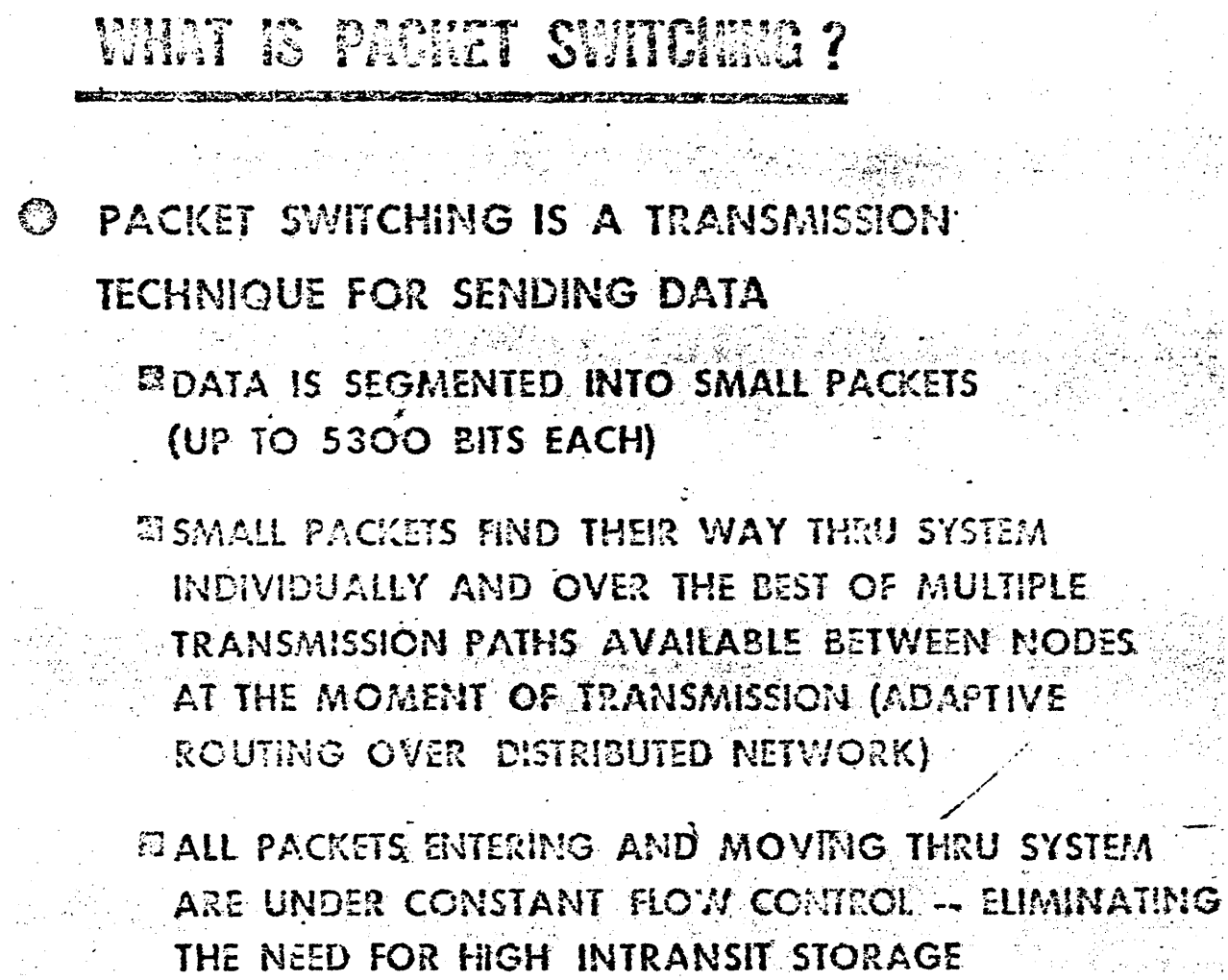


Chart VII

achievement of adaptive routing over a distributed network. With all packets entering the system and moving under constant flow control, therefore, the need for intransit storage is eliminated.

To the maximum extent possible (Chart XIII), the AUTODIN II specification and performance recognizes the technological advancements which evolved over many years from the development and operation of the Defense Department's ARPA network. While the AUTODIN II System is based on this technology, the operational characteristics and the operating performance varies greatly as shown on the following chart. Unavailable within the ARPA technology were the requirements for multilevel classified and precedence type traffic, high throughput with tremendous surge requirements in case of national emergencies and an unprecedented 12 percent growth factor each year through the life of the system.

REQUIREMENTS SPECIFICATION

Based upon a series of studies, the DOD developed the following list of primary requirements:

1. Phase I network configuration of four nodes, network control center and test facility, with capacity for 200 subscribers with 13 subsystems.
2. Modular growth capability to eight or more nodes with some 1500 subscribers operating in 47 subsystems.
3. Node capacities:
   - 150-200 full duplex line terminations.
   - Each line can be 110 bits per second to 56 kilobits per second (KBPS) for subscribers and 9.6 KBPS to 230 KBPS for trunks.
   - Throughput form 300 KBPS to 2.5 MBPS.

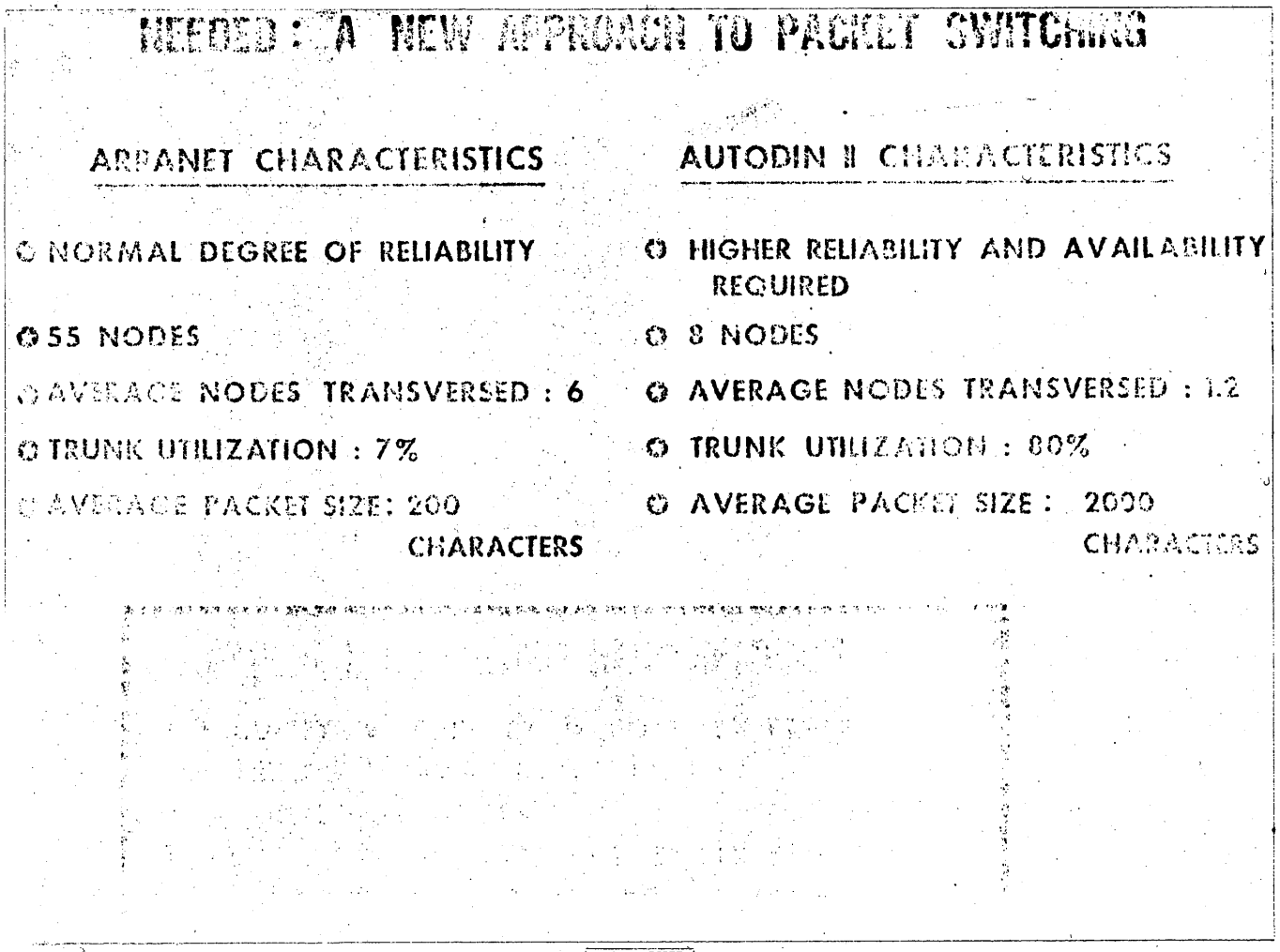


Chart VIII

4. Maximum packet size—5300 bits.
5. 16-level precedence system.
6. Non-blocking for high precedence interactive traffic.
7. Availability of 99 percent for single-homed and 99.5 percent for dual-homed subscribers.
8. Minimum impact on existing subscriber equipment software system.
9. End-to-end undetected error rate of less than 1×10^{-12}
10. Segment misdelivery rate of less than 1×10^{-11}
11. 300 ms. max backbone delay for interactive and query response, based on 600 bit packets.
12. Transparent to text.
13. Capable of being certified to handle all levels of classified traffic.

NETWORK DESIGN

The Network Design was predicated on the following Government provided projected subscriber data on the present AUTODIN I System and the future system:

TRAFFIC STATISTICS (BUSY HOUR)

Average Traffic Volume, including AUTODIN I	7.2×10**9 Bits
AUTODIN I Traffic Volume Trunk	4.9×10**8 Bits
Peak Traffic Volume Originated in any one second	8.1×10**6 Bits
Computer Traffic Volume	5.9×10**9 Bits
Terminal Traffic Volume	8.5×10**8 Bits

Based upon these and other data, a network design of eight packet switching nodes (PSNs) located at the eight AUTODIN I sites in the Continental U.S. was developed. Backbone trunks between these nodes will operate at 56 KBPS with sufficient connectivity to provide at least three routes out of each node.

Chart IX shows the trunk speeds between DIN II sites and between AUTODIN II and AUTODIN I sites.

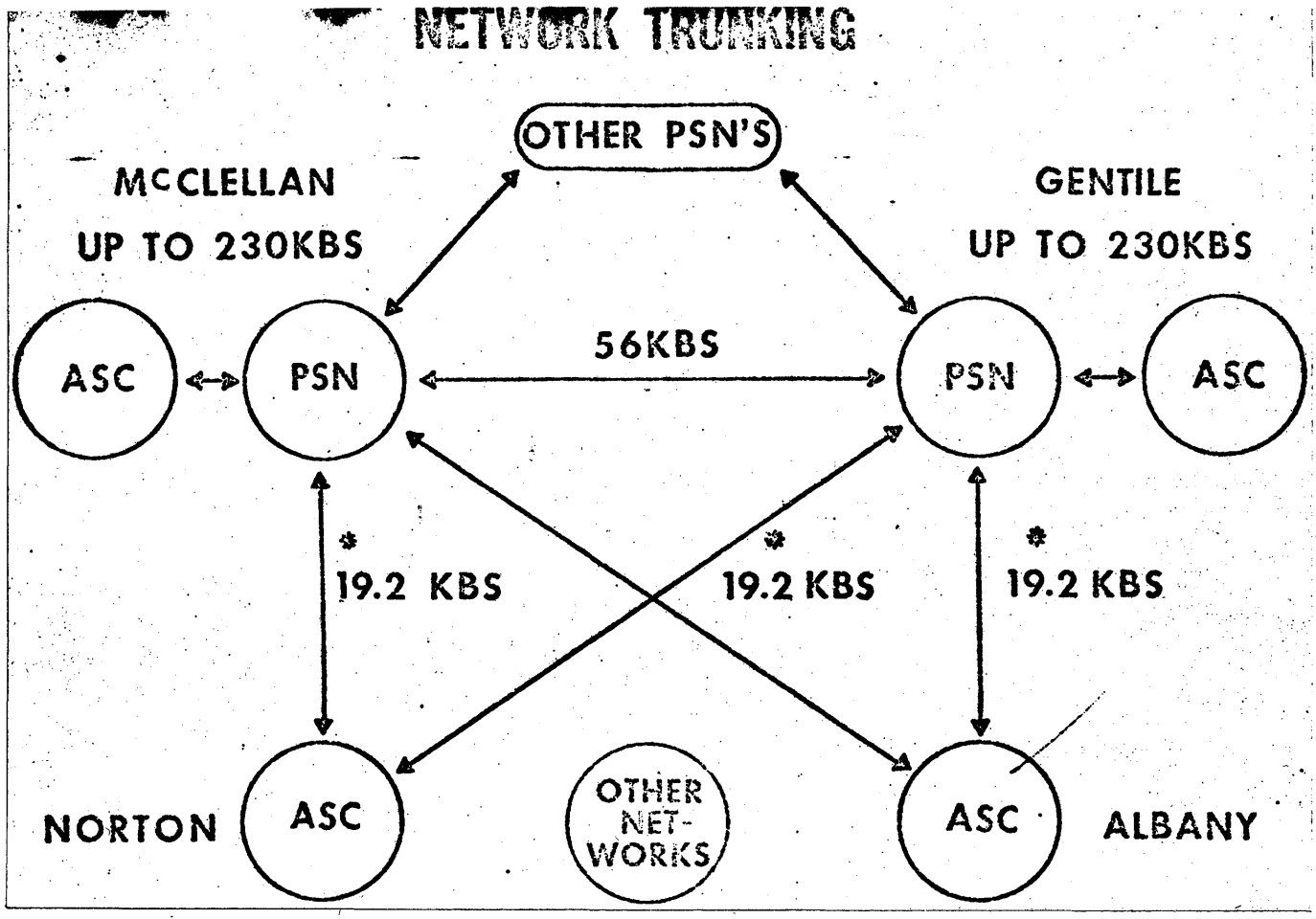


Chart IX

The initial network is scheduled for cutover in January 1979 with three nodes at Fort Detrick, Frederick, Maryland; Tinker Air Force Base, Oklahoma City, Oklahoma; and McClellan Air Force Base, Sacramento, California. After system testing, a fourth node will be added at Gentle Air Force Base, Dayton, Ohio in April 1979. The system capacity can be increased by modularly expanding the four initial nodes and by adding additional nodes.

It is anticipated that by 1981 traffic will have increased to require four additional nodes to be located at the Hancock Air Force Base, Syracuse, New York; Marine Corp Base, Albany, Georgia; Andrews Air Force Base, Maryland; Norton Air Force Base, California. At this point in the implementation plan, all CONUS AUTODIN I sites, except Hawaii, will have a colocated DIN II switching center.

SECURITY REQUIREMENTS

The overall AUTODIN II security requirement is to provide a system which does not allow unauthorized acquisition or alteration of classified information.

This is accomplished by a combination of techniques and procedures which include personnel security, physical security, administrative security, communications security, emanations security and software security.

The first five of these are well developed and well-known to military communicators. Software Security, however, is not so well developed and considerable research is still going on in this area. In AUTODIN II, software security will be achieved by use of security kernel technology derived from the MITRE model of DOD security regulations. This approach integrates all of the security controls for the system into a small portion of the operating system code which can be verified to operate correctly and to be unalterable by any other part of the system.

The basic elements are summarized as follows:

1. *PACKET SWITCHING NODE (PSN)*

   (a) Switch Control Module (SCM)
   (b) Line Control Module (LCM)

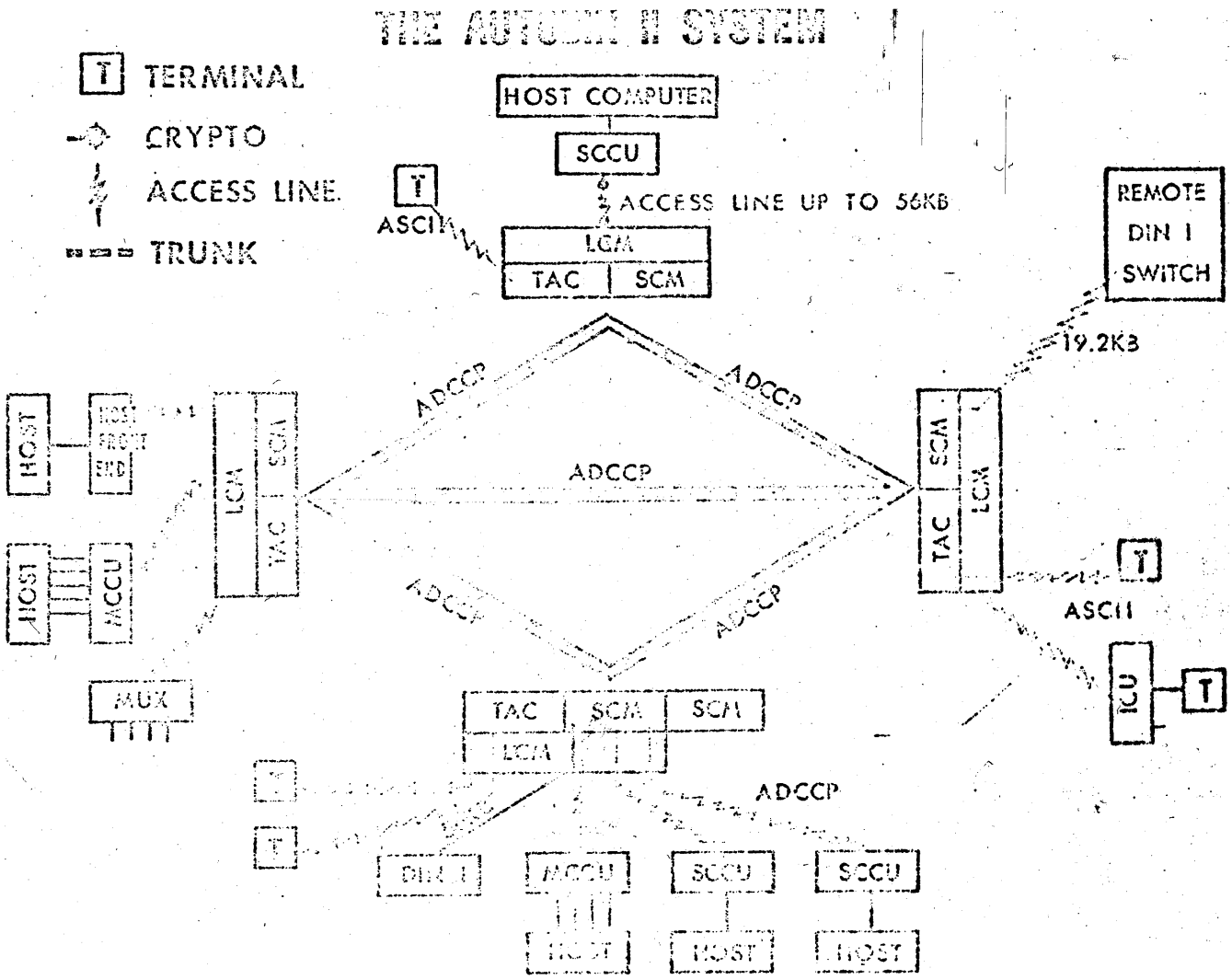


Chart X

2. *TRANSPARENT TERMINAL INTERFACE*
   (a) Terminal Access Controller (TAC)
   (b) Interface Control Unit (ICU)
3. *TRANSPARENT HOST INTERFACE*
   (a) Single Channel Control Unit (SCCU)
   (b) Multiple Channel Control Unit (MCCU)
4. *TRANSMISSION NETWORK*
5. *NETWORK ENCRYPTION EQUIPMENT*
   (a) Crypto Auxiliary Unit (CAU)
   (b) KG-13
6. *TEST FACILITIES AT EACH CENTER*
7. *CENTRAL NETWORK CONTROL CENTER AT DCA HEADQUARTERS*

Chart X highlights, in an overall system configuration, most of the above-mentioned network elements. The distributed network, using 56 KBS trunks initially interconnects all four of the initial system nodes directly providing the adapted routing principle to this packet switching network. A colocated DIN I facility illustrated in the southern switching node is directly interconnected by a high speed local trunk, while a remote AUTODIN I switch is terminated into the eastern node by a 19.2 KBS interstate transmission circuit. The chart also illustrates the use of the various host interface devices that may eliminate the need for reprogramming directly in the various host computers interconnect into the AUTODIN II System. Single-channel (1) and Multichannel (32) Control Units are illustrated at each of the four packet switching nodes. The SCCU and MCCUs are located on customer premises directly associated with the host computers.

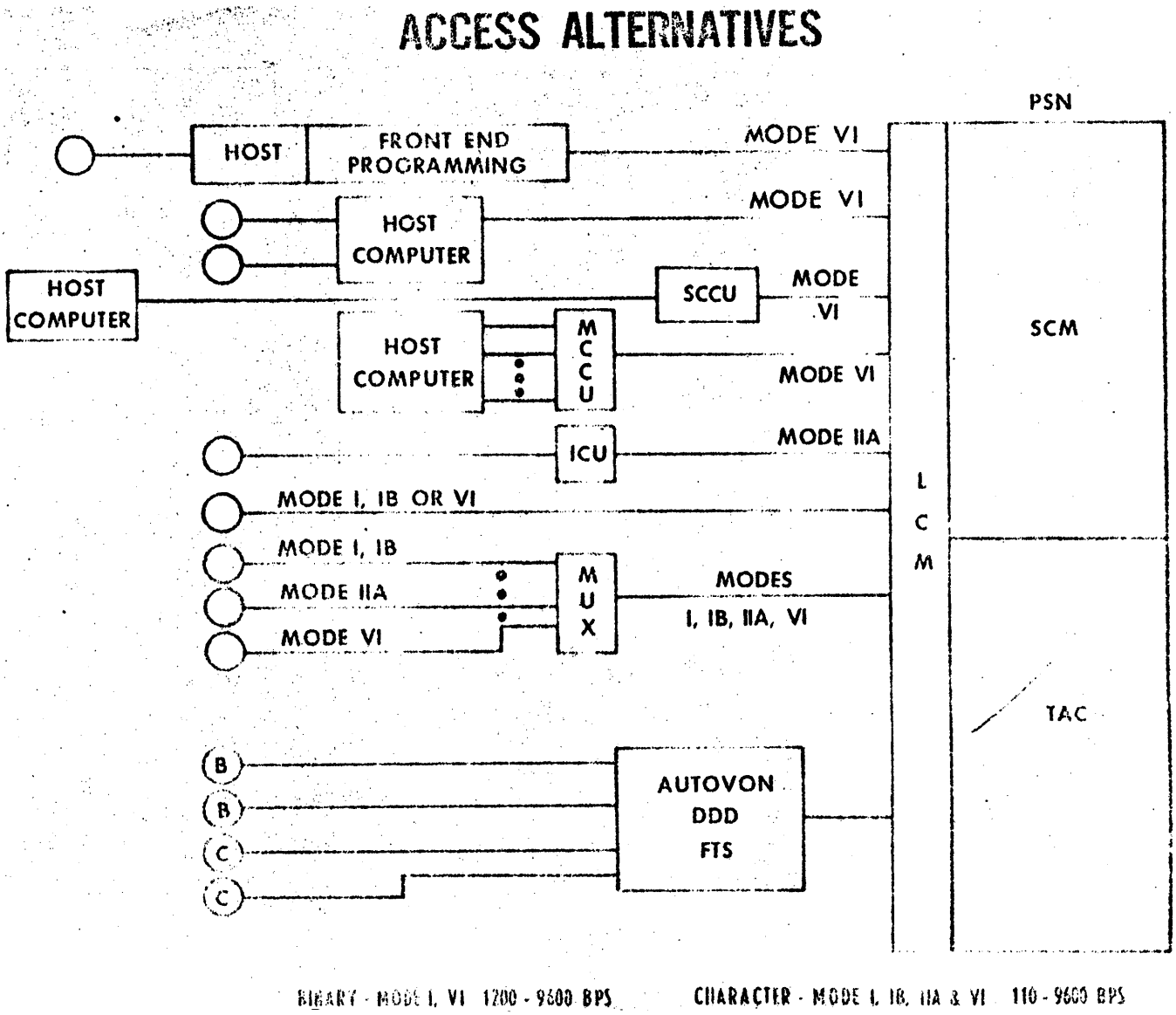


Chart XI

The system also indicates that terminal devices connected to the system go directly to the Terminal Access Controller (TAC). The TAC, however, is located within each node for one TAC unit can, in fact, handle 250 individual remote terminal devices. An Interconnect Control Unit (ICU) can be provided, if necessary, when terminals do not themselves have direct interconnectability.

Chart XI highlights in more detail the various means to access the system. Host computers can provide the interconnect protocol himself by reprogramming the host computer or adding a front-end device, or more simply they can employ either of the Western Union furnished Channel Control Units (single or multichannel) and avoid any local software modification. Shown also are the various terminal modes terminating directly into the TAC, either directly or through the ICU device. Also illustrated is the ability of the system to terminate circuits utilizing the DDDs, the Government AUTOVON, or the Federal Telephone System.

Chart XII illustrates the makeup of the packet, which is distributed throughout the network. The packet is 5300 bits in length with approximately 600 bits devoted to overhead and 4700 bits reserved for the transmission of data. The host information coming through the Channel Control Units and terminal information coming through the Terminal Access

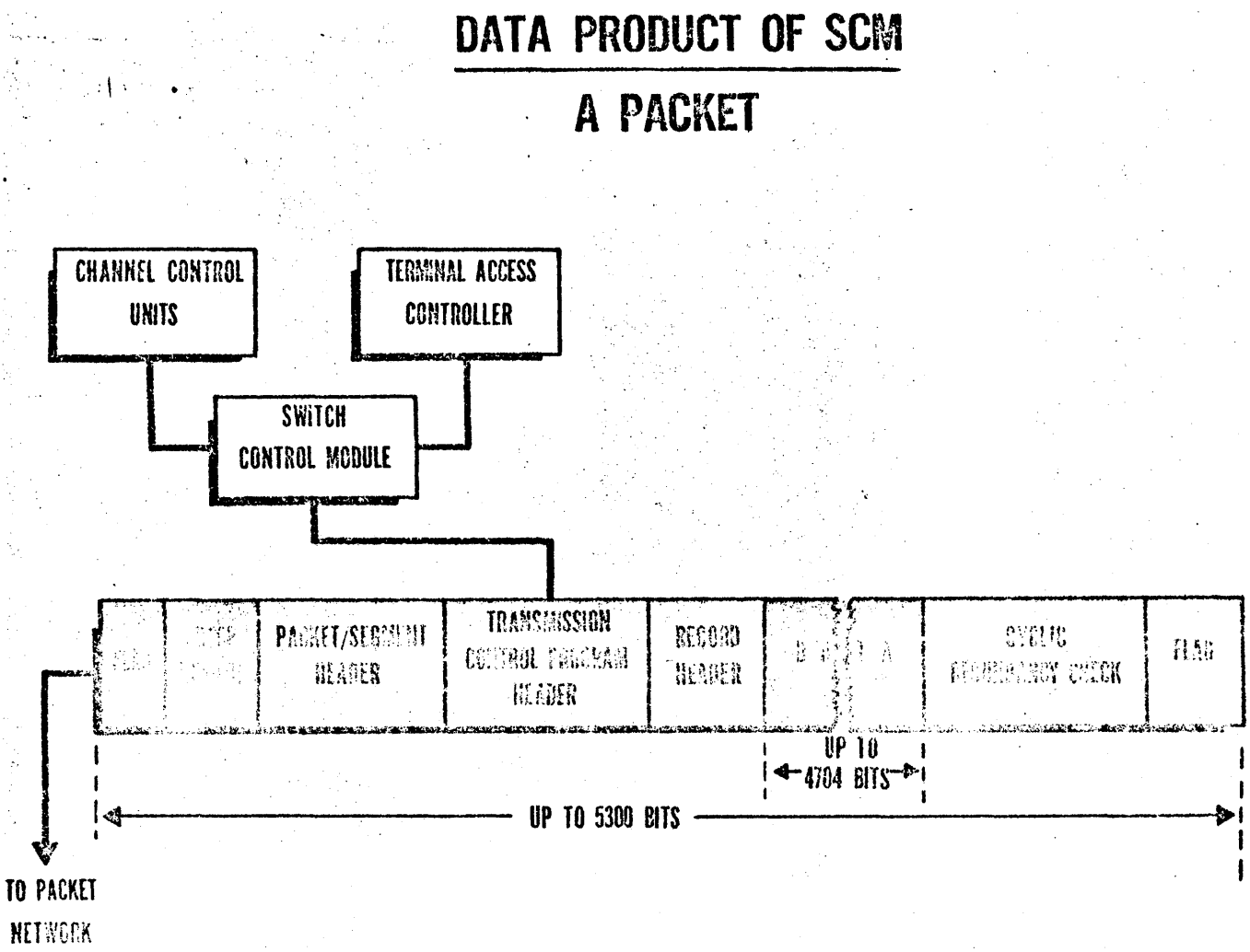


Chart XII

Controllers are first converted to segments in these two units and then converted to packets in the Switch Control Module just before entering the transmission system.

NETWORK CONTROL CENTER (NCC)

A Network Control Center monitors the operational status of the circuits, switching computers and user computer. It will assist in the diagnosing and correction of malfunctions. The NCC will achieve this by using the same DEC PPD 11/70 Processor as is used in the PSN. This PDP 11/70 will be complemented with magnetic tape and disc storage, high speed printers and four CRT positions for operator interface. It will be connected to at least one PSN and possibly two for redundancy connection will take place via communication links.

PATCH AND TEST FACILITY SUBSYSTEM

The second segment of the PSN is the Patch and Test Facility. At the AUTODIN I colocated sites, the AUTODIN II PTF will be integrated with the existing AUTODIN I PTF. In these combined facilities monitoring function will be performed by the SCM. All monitor points that have a direct bearing on circuit reliability and quality will be automatically scanned, and the scan results sent to the SCM. All control commands required to restore or maintain opera-

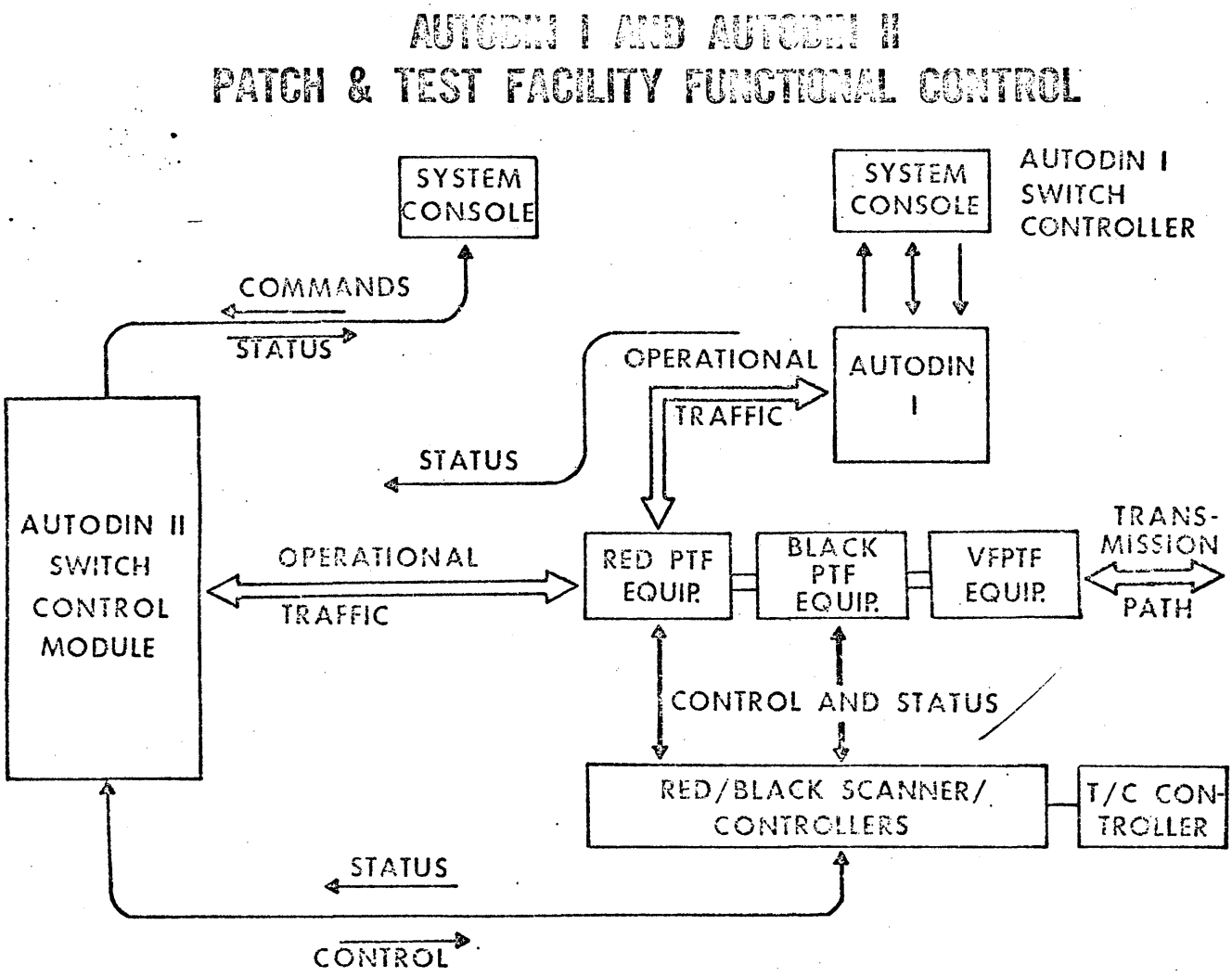


Chart XIII

tional circuit integrity will be formatted and transmitted by the SCM (either automatically or upon request from the tech controller) and will be executed by the PTF equipment. (Chart XIII)

CONCLUSIONS

Packet switching technology as added to the AUTODIN provides an example of the adaptation of new technology to an established environment. The user of this and the earlier implemented message switch technologies now has an opportunity to employ that unique capability that best satisfied his data communications need.

With the interconnect capability to be established between the AUTODIN I and AUTODIN II, the user can select that optimal network which meets his immediate need.

REFERENCES

1. Stathopoulos, A. and H. F. Caley, "The AUTODIN II Network," September 1977.
2. Owings, J., "The AUTODIN I Message Switching System," October 1962.
3. O'Rourke, D., "Message Processing—How The USAF Will Do It In AUTODIN," April 1961.
4. Wood, D. C., "Packet Switching Networks, An Economic Survey," August 1974.
5. Paz, I., "Advances In Optimal Routing Through Computer Networks," August 1974.
6. Sutter, W., and D. O'Rourke, "The Application of Packet Switching To The Federal Government AUTODIN."

Implications of a national computer network for higher education and science research*

by NORMAN R. NIELSEN
SRI International
Menlo Park, California

and

RONALD SEGAL
EDUCOM
Princeton, New Jersey

INTRODUCTION

A national computer network linking educational and research institutions is frequently proposed as a means of making available specialized computing resources, minimizing duplicate software development, and making more efficient use of the nation's total computing resources. However, a number of economic, political, and organizational, rather than technical, issues must be resolved before such a network is constructed. A Network Simulation Project, supported by the National Science Foundation, has been investigating these issues. Now in its third year, the project has gathered and analyzed a variety of data contributed by representative institutions concerning their computing resources, user profiles, and computing policies and practices.

Results of the modeling and data analysis support the arguments for operational economy, software development and maintenance economies, specialized service availability, and other proposed benefits of resource sharing at a national level. On the other hand, a number of barriers have been identified that must be addressed. These include institutional constraints on balance of payments (both positive and negative), institutional inabilities to expand service offerings to the network, difficulty of account control, and several network organizational and management problems.

The results of simulation and gaming exercises involving institutional administrators and other personnel were obtained too late for inclusion in the proceedings. These will be discussed at the presentation.

THE PROBLEM

Decision makers at educational and research institutions are grappling with increasingly difficult questions concerning the best means to satisfy the computing requirements of students, faculty, and staff members. The computing needs of these groups are increasing in variety and complexity, as well as in magnitude; and the means of satisfying these needs are becoming broader. Consequently, it is clear that attempting to meet all computing requirements "in-house" is no longer feasible. Technological advances in computing hardware will not, in themselves, solve the problem, for software and data base development, maintenance, and user support represent an increasingly larger part of the total cost of computing. As the variety of computing requirements increases, the non-hardware portion of computing costs can only escalate.

One mechanism for alleviating some of the problems posed is the computer network. Not only can networks provide ready access to a wide variety of available and supported software, but they can make these resources available very cost effective.[1] In 1972-73 EDUCOM,* with the financial support of the National Science Foundation, organized a series of General Working Seminars[2] on the subject of computer networking in higher education and research. Participants included university administrators, computer center directors, users from key disciplines, and computer scientists. Both general and specialized working sessions were organized to investigate many aspects of networking.

In session after session, regardless of their professional roles, participants came to similar general conclusions—namely, that it is now technically feasible to create a network linking computer facilities at colleges, universities, and research institutions around the country. Although many technological problems remain, the primary difficulties confronting such a network were felt to be non-technical in nature, involving economic, political, and organizational

* This work is sponsored by the National Science Foundation under Grant Number MCS75-03634, "A Simulation and Gaming Project for Inter-Institutional Computer Networking" (Network Simulation Project).

* EDUCOM is a consortium of more than 250 colleges, universities, and non-profit organizations that serve higher education. It was founded to help its members make the most effective use of computer and communications technology.

considerations. Obtaining a clear understanding of these factors before undertaking any large-scale networking development was seen as critical. It was this judgment that led to the initiation of the current research project.

THE RESEARCH PROJECT

While existing networks provide an indication of the costs and technical characteristics that might be associated with a national network, they cannot indicate how member institutions would be affected by economic, organizational, political, and intellectual considerations; how users would be affected by the availability of multiple resources at a variety of prices; how "balance of payments" problems (both deficits and surpluses) would impact institutions; and how a network might be impacted by institutional actions.

No existing network has the necessary characteristics to permit the study of these issues. Although the ARPA network, for example, offers interconnection between computing facilities at many universities and research institutions throughout the country, the single-source nature of the funding for most of the computing activities removes the source of many of the problems that need to be investigated.

The use of an actual network to examine many of these issues poses a number of other problems. Experimentation on a national basis would be extremely costly, would take several years to conduct, would be severely restrictive in the approaches and alternatives that could be investigated, would be disruptive to the normal operations of both the network and its institutions, and would require an inordinate commitment of energy and personal time from key individuals across the country.

In recognition of these difficulties, a simulation apporach was selected to investigate some of the behavioral and policy issues of such a computing network. Focus is on the potential impact of the network upon member institutions, as well as the impacts of institutional decisions and policies upon the network. The simulation approach provides flexibility as to the types of networking situations that can be examined, and allows for the testing, at a relatively low cost, of a variety of alternatives. For example, the model can be used as an analysis tool to investigate the impacts of a wide variety of behavioral assumptions. It can also support a game, played by institutional representatives, to investigate questions pertaining to likely behavior encountered in a networking environment.

Initially, a six month planning study was conducted by eight individuals knowledgeable in the areas of model building, gaming, economics, resource administration, and educational and research computing.[3] The present Network Simulation Project is a three-year project implementing that plan.

The first phase of the project was devoted to basic data collection and analysis. Each of eighteen participating institutions contributed data on their computing resources (hardware and software), pricing schedules, priorities, turnaround levels, demand, and user behavior profiles. The network simulation model was developed and used with the accumulated data base to conduct a series of preliminary investigations. Details of the phase one model development and findings have been reported in References 4 through 10.

The second phase of the project was devoted to including computing policy information in the institutional data base, and to refining the simulation model. Visiting teams interviewed administrators, users, and computer center personnel at each participating institution. The computing policies and practices in effect at each institution were deduced from this information and subsequently confirmed with the participants. This updated data base is currently being used in a series of further experiments using the network simulation model. The data has also been used to support many of the analyses discussed below.

The third phase of the project, currently in progress, has converted the simulation model into a gaming version with a conversational interface. Using this version, both a remote and a centralized set of gaming experiments are being conducted. Personnel from each of the eighteen institutions are actively participating, each representing the "best" interests of their home organization. In the remote experiments, each institution is operating with a "straw network," a static network using programmed decision rules for all institutions (except for the player's own institution). This provides for controlled experiments in which responses to particular types of circumstances and environmental conditions can be tested. The centralized experiments will then be conducted during an intensive live three-day session, which will permit the investigation of dynamic inter-institutional interactions.*

BENEFITS OF A NATIONAL NETWORK

Analyses of the resources and policy data collected from the participating institutions, as well as investigations using the network simulation model, have revealed a variety of positive factors associated with a national network. Several of these are summarized here.

Operational economies

The availability of network computing resources can lead to the combining of facilities, a reduction in size or range of offerings at some computer centers, and better utilization of the remaining facilities. Any of these alternatives can result in operational economies. For example, Harvard has been able to significantly reduce the size of its central computer facility, relying heavily on remote services obtained via telecommunications access. Similarly, Harvard's primary supplier, MIT, was able to justify an expanded and better staffed operation because of the guaranteed outside usage (income) from Harvard. This results in a more powerful and cost effective service to the MIT community. Obviously, a national network will not enable everyone to participate in this

* These experiments will have been completed by the NCC, and the results will be discussed during the authors' presentation at the Conference.

manner. However, the opportunity is there for institutions in appropriate circumstances.

Software and data base economies of scale

For a variety of services, software and data base maintenance costs—including associated user services such as documentation and support—do not vary significantly with usage. Since a centralized center permits spreading these costs over a larger number of users, it can provide substantial economies. As hardware costs continue to decline, and software-related costs continue to increase, these economies will become increasingly important.

Availability of specialized services

One of the great advantages of a network is that it allows specialization while still exploiting economies of scale. A computer center that offers a specialized service can attract enough network users to justify the high fixed costs of maintaining the service. Thus, the availability of such services is likely to increase rapidly. Networks can, in this way, complement localized centers. A computer center can choose to provide relatively standard services for its own users, plus a few specialized services for the national market, and then look to the network to obtain specialized services that it does not provide.

Data collected by the project indicate that there is a vast array of specialized services already available that are appropriate for a network environment. However, it is likely that these services only represent a small fraction of the total that would be developed once the potential usage base was realized.

Economies of comparative hardware advantage

Not every computer system can process every type of job with the same relative efficiency. Hardware and operating system differences, as well as local modifications to computer systems, have resulted in a wide variety of configurations, each having different processing characteristics and capabilities. As a consequence, systems have different relative advantages. If one were to take the processing that is being performed at a set of institutions, and redistribute it among these same institutions on the basis of the comparative advantage of their computer systems, net computer resource usage would decrease relative to capacity.

The EDUCOM Planning Council* [11] has developed a series of benchmark programs that illustrate the point dramatically.[12] Although subject to all the usual problems of "representativeness," the benchmarks have been run at each of the participating institutions. They provide a system-by-system comparison of the difficulty of converting a program from one site to another, the computer resources required to execute the programs, and the cost to execute the programs.

In the tests, there were several examples of price differences on the order of 10:1 and even 20:1 for the processing of a given job. Although the lowest priced facility is not always the most effective when all factors (e.g., turnaround and user support) are considered, these price differences are attractive enough to encourage individual users to take advantage of the comparative advantages of different systems, thus resulting in major economies. Admittedly, prices are not always directly tied to costs,[13] and many of the price differentials would diminish in a free market environment. However, the institutions involved believed that their pricing was an accurate reflection of true costs at their facility and that substantive changes would be unlikely.

The identities of the "lowest cost" and "highest cost" institutions were not constant across the benchmark jobs. That is, one institution was more efficient for some job classes but not for others. Thus, the trading opportunities are not one-sided and there should be a reasonable distribution of usage. This situation is analogous to the economies of comparative advantage in international trade.[14]

National software marketplace

At the present time there is no simple marketing or distribution mechanism for available software. To market a product, the developer must prepare user documentation and installation instructions, must prepare versions for different computer systems, must be prepared to consult on implementation problems, and must develop and distribute maintenance updates for all systems. For the person or organization that is not in the business of software development, this is a significant barrier. Thus, most local developments are not exported, forcing duplicate development if the capability is to be obtained at all.

The existence of a national network would change much of this, since prospective users would be able to use the software on the developer's own system. All users would run with the same program on the same system, permitting a single set of documentation materials and a single maintenance operation. Thus, the burdens upon the developer (who is often more interested in his discipline or research subject than he is in software development and marketing) are greatly reduced.

A national market would also make it possible to apply usage royalties. Not only could this be used to provide an economic incentive to develop and support good software, but it could also provide a mechanism to support the additional costs of polished documentation and user support. Thus, the network framework could support greater sharing of software development and a faster spread of beneficial software.

* The Planning Council is a cooperative effort of twenty-two universities that have decided jointly to "determine how the computing needs of their institutions can best be met, assess the means of achieving efficient and effective resource sharing, and develop a national 'facilitating network' linking computers at colleges and universities throughout the United States."

Management improvements

Although computing has grown into a substantial business at many institutions, the effectiveness of the management and operation of the local computer center has not always developed analogously. In all too many instances, our data and on-site interviews revealed that large computational facilities are being run as if they were small service facilities (which indeed they were not very many years ago). Yet large computing operations are a business and must be recognized and operated as such.

The existence of a national network would bring a measure of comparison and competition to all participating institutions. Not only would there be price comparisons by prospective users, but there would also be service, support, and reliability comparisons as well. Poorly managed facilities would find themselves in a poor competitive position. If local administrations permit free choice, such facilities would be forced to either upgrade operations or lose business to the better-run facilities. From the viewpoint of many individual facilities, this would be a traumatic experience. In any event, though, the user will perceive more effective service.

Positive attitude

A positive attitude is critical to the success of any endeavor. Surveys of regional networks in higher education[15,16] showed that many networks succeeded (or failed) in large part becuase of the beliefs of key participants that the network would succeed (or fail). In the case of a possible national network, our interviews with administrators, computer center directors, and others revealed a consensus of positive attitudes. Administrators envisioned cost savings, increased research potential, and better overall service to their community. Computer center personnel anticipated a wider audience for their offerings, greater service to internal users, and elimination of some costly local services. Users felt that "competition" would eventually bring more cost effective service to them, and that the expanded availability of suppliers and offerings would be beneficial. Although such attitudes do not, of course, guarantee the success of a networking activity, they are a vital ingredient.

Preliminary model results

A number of explorations have already been conducted using the basic simulation model.[4,9] For these tests, perceived decision and behavior patterns of all sites were preprogrammed (as compared to the gaming studies in which real people will be making decisions at each site). Although results are not yet conclusive or comprehensive, indications are positive in all areas tested. The network appeared stable in the face of large shocks; there is the opportunity for large cost savings to users; communications costs are relatively low for many of the likely early usage categories; the more "attractive" supplier sites seem to have sufficient available capacity; and the different hardware configurations appear to present useful differences in performance features. More sophisticated and revealing research experiments, as described in a later section, are now being conducted.

BARRIERS TO NETWORKING

Our analyses to date have also revealed a number of barriers to networking. Although adverse, these barriers are not insurmountable. However, they must be addressed by any network undertaking.

Institutions are unprepared to modify computer facilities

Many institutions are viewing the network as a mechanism for selling "a little" excess capacity and for obtaining "a little" extra revenue. They are not prepared to serve the growing computing requirements of a national user community. That is, most institutions, even if they had a highly sought after set of services, would be very unlikely to install additional large-scale computing equipment solely to serve the needs of network users. In many cases, the charter of the institution or the laws or policies under which it operates preclude such an expansion.

On the other hand, most institutions believe (in most cases, quite strongly) that they must have a major central facility on site if they are to maintain their research and educational position. Further, few institutions have a great deal of short-term flexibility to reduce their scale of computer operations, even if they deemed it advisable. There are purchased equipment, long-term leases, personnel commitments, state restrictions, and other considerations that make rapid change difficult.

Thus, we have a situation in which facilities are unlikely to be expanded or contracted significantly. This implies that the imports and the exports of computer services at a site must either be fairly closely matched, or must be small in magnitude relative to the total computing budget. We have made several simulation runs using a policy of "no imbalance of payments" on the part of each institution. The results do not bode well for network vitality, for the work flows that can develop under these assumptions are rather limited.

Lack of incentives for sharing

One of the serious problems that is likely to constrain the growth of network computing is the lack of strong incentives to encourage sharing. Incentives are missing for both the buyer and the seller of services, and at the level of both the individual and the institution.

On the buying side, individual users are likely to be frustrated by shortcomings in documentation, consulting, and other support services; by the unreliability of existing software and data bases; and by difficulties in locating and accessing appropriate remote resources. At the institutional

level, money spent on remote services by internal constituents is usually perceived as "real" money resulting in a financial drain to the institution; whereas the cost of local services is largely fixed and does not impose significant incremental costs in the short run if usage is increased (nor incremental savings if it is reduced). Consequently, there is often a reluctance to permit access to outside services.

Unfortunately, the situation is just as difficult on the supplying side. Faculty members and other individuals who create programs and data bases typically receive neither financial benefit, nor professional rewards such as those associated with the publication of a paper in a prestige journal. Consequently, there is little incentive to undertake the incremental work required to document, remove errors, and otherwise convert a successful "local" resource into one that is acceptable for a network environment.

Similarly, institutions lack strong incentives as suppliers. Economies of scale in computer hardware have traditionally been one of the primary justifications for concentrating a variety of computing functions into a single large central processor. However, except for a few quite specialized cases, hardware economies are no longer an important motivation. Minicomputers now compete economically with the large machines for raw cycles, and any significant economies of scale for very large jobs can usually be obtained regionally, if not locally. Further, the cost of the central processor as a fraction of total computing costs is shrinking rapidly. However, economies of scale still hold for large, specialized processors in such fields as physics and meteorology. Thus, for example, the ILLIAC IV computer at NASA Ames is likely to become a major network resource. Most institutions, therefore, have little to gain from the use of network demand to justify a larger central processor.

Unfair competition

Not all institutions that might belong to the network are bound to operate under the same pricing regulations. For example, most educational and non-profit research institutions performing government-funded research are bound to operate all services on a fully cost-recovered basis. On the other hand, government laboratories often acquire computing equipment with capital funds which do not appear in their cost base. Thus, they can and do offer services to other users at marginal or operational cost. Further, commercial organizations could offer services via the same or a compatible network. Loss leaders and other types of promotional prices could be established. Thus, institutions committed to a large central facility could find their user base severely eroded by price competition, their ability to compete on price severely restricted, and their ability to reduce expenses limited.

Guarantees

Institutions that might potentially offer computer services over a national network have shown little willingness to provide service guarantees to outside users, except in some very special cases. Their basic operating policy is "our users first" (and properly so according to their charters). Thus, if demand were to exceed available capacity, the local user would be served and the network user would often be denied service. This means that other institutions and users cannot depend upon the continued availability of service. Although it is unlikely that abrupt large-scale cut-offs would ever occur, the possibility of disruption is a significant concern.

Looking at the problem the other way, most users are not in a position to make long-term usage commitments. However, without such commitments, most supplier installations are unwilling to assume the risk inherent in expanding capacity or services.

Usage barriers

While price (economics) is a powerful motivator with respect to users shopping for and switching to outside services, user behavior profiles developed at each institution confirm that current computing technology limitations are significant. First, there are differences in command languages, file procedures, accounting methods, and the like from site to site. Even such things as "standard" FORTRAN compilers have differences and unique "quirks." Learning to work with many different sites is such a personal burden for most users that only a small number of them are even potential network customers.

Second, software conversion is time-consuming and "unrewarding." Once a site is selected as the supplier, the software will likely be developed and then executed at that site for the duration of its useful life. Evidence indicates that even sharp differences in price or service levels are unlikely to cause any appreciable shifting of existing programs between sites.

A third barrier is information. The user needs to find out "what is offered where," how the system operates, what the pricing structure is, and so forth. Further, he needs a mechanism for determining how effective each site is for his own computing needs. Unless this fairly costly process can be written off over a significant amount of future computing, it is not cost-effective to consider many other supplier sites. To the extent that there is no central organization to collect, keep current, and distribute such information, many users will be unable to find alternate computing sites that would be beneficial.

Network overhead

Participation in a network is not without cost. In addition to the base cost of communications between sites, interfaces to the network communications facilities have to be implemented for each host computer; billing and accounting routines have to be modified so that this information can be provided to each user, institution, or network clearing house as appropriate; administrative structures must be implemented; and standards and interface procedures must be

provided and enforced. For an institution that only wants to distribute or gain access to a small volume of services, these requirements could be formidable.

Security

Current systems provide inadequate protection against malicious or unintentional access to system resources. Until this situation is corrected, most institutions will choose not to maintain sensitive data on remote storage devices accessible via a network. Applications involving personnel or student records, detailed financial data, or other confidential information will thus be unlikely candidates for early network service.

Another aspect of security is protection against unauthorized use of network resources. When an institution provides its own computing services, its total financial risk is limited to the expenses of running the computer center. In the case of a network, however, an institution's potential consumption of resources could be virtually unlimited.

Government, regulatory, and tax issues

There are a number of unresolved legal and regulatory issues that must be faced. For example, can a nonprofit educational institution freely sell computing services and still retain its nonprofit status? How will the federal government requirement that it receive the most favorable price offered to any buyer impact the ability of institutions to use flexible prices in rationing their resources more efficiently? Will the constraint imposed by many state legislatures that both the sale and purchase of computing resources at state universities be limited to in-state facilities exclude these institutions from national network participation?

Many institutions, because of educational discounts associated with equipment acquisitions, institutional charters, or tax regulations, are unable to process particular types of work. Commerical work, for example, is prohibited or severely limited at many sites. Although an institution can exercise control to some degree over its own local users (i.e., forbid commercial accounts), it is very difficult to exercise such control over unknown network users. Thus, a commercial user that can't use the low-cost service at institution "X", might be able to obtain system entry at less restrictive institution "Y" and hence be able to access institution "X" as a "Y-user." This is a serious concern to some institutions.

Unfulfilled expectations

While many administrators, computer center directors, and others have very favorable attitudes toward networking (as mentioned earlier), there will be many disappointments due to inappropriate expectations. For example, our data show that most institutions believe that they offer attractive, economical services and that, in a free market environment, they would be net sellers to the network. It is not the position of this research to critique the individual viewpoints; however, it is obvious that not all institutions can be net sellers. On numerous occasions in the regional networking programs undertaken in the late 1960's,[15] such disappointments resulted in changes in attitudes and the erection of artificial barriers to trade that adversely affected the networking endeavor.

LIKELY CHARACTERISTICS OF A NATIONAL NETWORK

Although our research has not been completed, a number of indications have been found as to what the likely characteristics of a national network might be. Some of the areas being investigated more thoroughly are listed below.

Network development

1. Usage Growth—Network usage is likely to increase at a steady rate without any sudden surges in growth. A relatively long transition to a network environment is expected for buyers to gain experience with network services, for institutions to adjust their internal capacities in conformity with changes in demand patterns, and for suppliers to develop more powerful aids for assisting remote users.
2. Evolution—The network is likely to evolve gradually, rather than be implemented in one major step. To adapt to changing needs and technology, the design at each stage of implementation should be kept as open-ended as possible and should avoid large fixed costs (e.g., costs should be incurred in proportion to the level of network activity).
3. Volume of Usage—Among participants there will be a wide variance in network activity, with individual sites ranging from no use to total dependence. The most optimistic present estimates suggest that network dollar volume will represent less than five percent of the total computing done by member institutions. Note, however, that even at very low per unit levels, aggregate flows from all members could ultimately be very large.

Network characteristics

1. Hierarchy of Networks—It is likely that there will be a hierarchy of networks available to users, each serving a different need. These would include local mini-computer networks, campus centers, regional and state networks, national networks, and commercial facilities.
2. Autonomy of Members—Colleges and universities in this country are largely autonomous and show no intention of weakening this autonomy in their quest for computing efficiency or power. This strongly suggests that a successful national network for higher education must be largely decentralized. Each institution should be allowed to choose which services it will buy, which services it will sell, and to whom it will sell them. An institution should also be free to

set both the prices and the quality of services it provides to others.

3. Central Facilitating Function—Although decision making is likely to be largely decentralized, certain facilitating functions need to be provided centrally. Included in such functions are billing mechanisms, information on services and prices, standards, security procedures, and provision of network-wide user services.

4. Financially Self-Supporting—A national network of autonomous members must be self-supporting and will likely rely on a free market mechanism to provide incentives for both suppliers and buyers. Although financial aid from government agencies and private foundations might be available to accelerate the development process, the operational network must be economically viable on its own without subsidy.

5. Evolution of Successful Suppliers—The data collected to date indicate that many of the most popular sites will "spin-off" their computing facilities (or at least that portion serving external customers) to a separate, not-for-profit or even for-profit subsidiary. Under such a scenario, many universities and research institutes would reduce their own local computing capabilities and either use or provide service from a "computer service business." Consequently, what started as a network linking the computer facilities of a number of research and educational institutions, might well turn into a network linking a number of university-owned service bureaus. Several examples of such "spin-off" activities already exist. The best known are the Dartmouth Time Sharing System (DTSS), the Ohio College Library Center (OCLC), and the Stanford Library System (Ballots).

Network usage modes

1. Functionally Distributed—Economics favors local or regional computing for most services, with the national network providing specialized services that cannot be economically supplied or maintained locally. Most suppliers will therefore concentrate on providing a relatively few specialized services that exploit their comparative advantage.

2. Specialized Services—The specific specialized services that are likely to exist in a network environment are difficult to predict in advance, although communications constraints are currently forcing an emphasis on services with low to medium volumes of data transfer. Candidate specialized services will be those that require some combination of complex software, large data bases, sophisticated support, or specialized facilities. Examples include library services, bibliographic services, person-to-person communications ("electronic mail"), chemistry and census data bases, econometric forecasting systems, and high energy physics computations.

User support

One of the barriers to network usage mentioned earlier concerned the difficulty of appropriately supporting remote users. In addition to all of the forms of user services currently provided, the network will likely also offer some combination of the following:

- Both local and central consultants who are knowledgeable about services available over the network, or at least can facilitate the acquisition of information.
- "Circuit riders" who periodically visit remote sites.
- Remote consulting services provided on-line over a "hot-line" or by means of an "electronic mailbox."
- High quality documentation designed to serve the independent remote user.
- Computer-assisted training aids.
- On-line information data bases to assist users in locating remote services appropriate to their needs.

CONFIRMATION EXPERIMENTS

The findings discussed above are based upon the resource and policy data that have been collected in the first two phases of the Network Simulation Project and upon the initial simulation experiments. In order to confirm the validity of these findings, a further series of experiments is currently under way.

The first set of experiments is being conducted by project staff using the basic network simulation model together with site-specific data bases and behavioral information. General topics such as the following are being investigated:[10]

- Expected network performance
- Site specialization
- Network stability
- Network resource-sharing potential
- Communications costs
- Service pricing policies
- Provision of special services
- Network equilibrium conditions
- Quality of network information provided
- Network growth effects

These investigations will continue in parallel with the participant experiments (see below).

The second series of tests (referred to as the "straw games") involves active participants from each institution playing against a programmed network ("programmed" is used in the sense that behavior patterns, decision rules, offerings, and so forth are pre-entered for all sites except one). This permits a series of experiments to be performed testing institutional reaction to controlled environmental circumstances. Typical experiments in this series involve:

- Reactions of supplier sites to increasing levels of demand from the network
- Latent demand for various proposed science information services
- Effects on supplier sites of large erratic demands from the network
- Shocks, such as major suppliers leaving the network unexpectedly

- Commercial vendors joining the network
- Reactions of supplier sites when they lose usage
- Effects of small schools attempting to satisfy much of their computing needs via the network.
- Implications of sites reducing their in-house facilities as network dependence increases.

The third series of experiments (the "live game") will be conducted at a central site with live players from each institution on the network playing simultaneously. This will permit a series of experiments to test inter- and intra-institutional interactions on a dynamic basis.

Typical experiments in this series involve:

- Institutional budget crises
- Unexpected federal edicts that affect the way many institutions have been using the network
- Changes in tax regulations for non-profit institutions
- Availability of a variety of competitive science information services
- Network "shocks"
- Technological breakthroughs in communications cost, capacity, interfaces, etc.
- Breaches of security
- Impact of changing network administrative structures

SUMMARY AND CONCLUSIONS

The studies to date support the expectation that establishment of a national resource-sharing computing network for higher education and science research is a formidable task. The barriers described in this paper will not be easily overcome. However, all evidence points to a potential high payoff from, if not a vital requirement for, such a network. In particular, the specialized hardware and software resources that would be accessible could not be made broadly available via any other mechanism.

Early indicators are extremely favorable. Both potential suppliers and potential users see it in their self-interest to participate in networking activities. Attitudes toward networking are positive and constructive, and administrators at the highest level are becoming actively involved. The likely deciding factor is whether or not short term pay-offs can be made attractive enough to sustain interest during the difficult development period.

Much research still remains to be done, and the efforts described in this paper are continuing. The various gaming studies are enabling decision makers at participating institutions to gain experience with operational considerations in a network environment. As their experience matures, they are becoming more confident in their ability to properly represent their institutional needs in such an environment. The above experiences are also permitting more general, long-term research into the design and impact issues relative to the evolution and operation of a national network.

REFERENCES

1. Emery, J. C., "Implementation of a Facilitating Network," Policies, Strategies and Plans for Computing in Higher Education, EDUCOM, Princeton, New Jersey, 1976.
2. Greenberger, M., J. Aronofsky, J. L. McKenney, and W. F. Massey, eds. *Networks for Research and Education: Sharing of Computer and Information Resources Nationwide,* MIT Press, Cambridge, Massachusetts, 1974.
3. Report to the National Science Foundation on the Study to Develop a Research Plan for a Simulation and Gaming Project for Inter-Institutional Networking, Grant GJ-41429, EDUCOM, Princeton, New Jersey, August 1974.
4. Report to the National Science Foundation of Year 1 of the Simulation and Gaming Project for Inter-Institutional Computer Networking. Grant DCR75-03634, EDUCOM, Princeton, New Jersey, July 1976. (NTIS Document Number PB259789/AS)
5. Segal, R. and N. White, "Representation of Workloads in a Network Environment," *Proceedings: Summer Computer Simulation Conference* Washington, D.C., July 1976.
6. Segal, R., and N. White, "Management of a Large Computer Network Simulation Project," *Proceedings: Fourth Annual Symposium on the Simulation of Computer Systems,* Boulder, Colorado, August 1976. (Also published in *Simuletter,* Journal of the ACM-SIGSIM, Volume 7, No. 4, July 1976.)
7. Segal, R., and N. White, "A Simulation Model of a National Computer Resource-Sharing Network for Education and Research," *Proceedings: Computer Networks Conference,* Gaithersburg, Maryland, November 1976.
8. Nielsen,N., and R. Segal, "Modeling an Inter-Institutional Computer Network," EDUCOM Bulletin of the Interuniversity Communications Council, Volume 11, No. 4, Winter, 1976/77.
9. Nielsen, N., and R. Segal, "Simulation of Institutional Behavior in a National Networking Environment," *Proceedings: Winter Simulation Conference,* Gaithersburg, Maryland, December 1976.
10. Emery, J. C., N. Nielsen, B. O'Neal, and R. Segal, A Simulation Model of a National Network for Education and Research," *Proceedings: 15th Annual AEDS Convention,* Fort Worth, Texas, April 1977.
11. Planning Council on Computing in Education and Research, EDUCOM, Princeton, New Jersey, May 1976.
12. Heller, P., Benchmarking the Price of Computing: Results of a Survey," *Computer Networks,* Vol. 1, No. 1, June 1976.
13. Bernard, D., et al., *Charging for Computer Services: Principles and Guidelines,* PBI Books, Petrocelli, New York, New York, 1977.
14. Berg, S. V., "Planning for Computer Networks: the Trade Analogy," *Management Science,* Vol. 21, No. 12, August 1975.
15. Weingarten, F., N. Nielsen, J. Whiteley, and G. Weeg, "A Study of Regional Computer Networks," University of Iowa Report, Iowa City, Iowa, 1973.
16. Nielsen, N., "The Implications of Star Computing Networks," *Proceedings of IFIP Congress 74,* North Holland Publishing Company, Amsterdam, 1974.

A comparison of network architectures— The ARPANET and SNA

by GILBERT FALK
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

The ARPANET is a packet-switched data network which facilitates resource sharing and supports distributed data processing. The ARPANET technology and its descendants have had a significant impact on the direction of data communications over the past eight years. More recently, IBM announced Systems Network Architecture (SNA) which is targeted at satisfying a similar set of requirements. SNA will undoubtedly also have a major impact on the future of data communications. This paper presents a comparison of these two network concepts by discussing architectural and protocol similarities and differences employed to support process-to-process level communication. The goal is both to provide useful information and to stimulate interest in the analysis of alternate architectural approaches.

The presentation that follows identifies key properties and functions of data networks in general and discusses briefly how they are performed within each of the two network architectures. An elementary knowledge of both the ARPANET and SNA is assumed. Information describing the design and implementation of the ARPANET can be found in References 1-3. A presentation of the concepts embodied within SNA can be found in References 4-8.

Architectural similarities in these two designs result from fundamental similarities of objective. Both network architectures have as primary goals the abilities to support distributed processing and to facilitate resource sharing. In the case of the ARPANET the motivation for providing these capabilities was the need to share large hardware and software resources and to establish a facility to support computer system research. The development of SNA by IBM, on the other hand, was largely driven by the economic need to take advantage of equipment based on LSI technology. This technology provides the capability to locate many previously centralized functions in terminals, device controllers, communication front-ends, etc. Both designs have attempted to provide a single general-purpose interprocess communication facility upon which application level communication rests. A central theme in the development of both architectures is the separation of the data processing and data communication functions.

Perhaps more important than the similarities of objective are a number of fundamental differences between the ARPANET and SNA environments. These environmental differences provide a partial explanation of the individual architectural differences that are present in the two designs. The ARPANET was developed for operation within a research environment. Within such an environment it is possible to experiment a bit in the design and operation of an evolving network. In a commercial environment, on the other hand, conservatism dictates more cautious development. While SNA has been developed as a new network architecture, it nevertheless has the requirement of accommodating a large existing base of installed software systems (e.g., IMS, CICS). ARPANET development, on the other hand, was not constrained in this way. ARPANET host software was developed for the most part independent of any existing application software within the hosts. SNA was designed to provide a networking capability for equipment from a single vendor. The ARPANET, on the other hand, was developed from the start to provide a networking capability for subscriber equipment from many different vendors. Finally, SNA and the ARPANET differ in the nature of the two network concepts. SNA is an implementation independent architecture. This architecture is realized differently within different IBM computer and communication products. The ARPANET, on the other hand, is an evolving implementation of the general concept of packet-switching.

COMPARISON OF NETWORK CONCEPTS AND MECHANISMS

The following subsections provide a side-by-side comparison of a number of aspects of the ARPANET and SNA network architectures.

Physical network concept

Figure 1 illustrates typical physical network configurations within SNA and the ARPANET. Figure 1a illustrates four distinct types of physical units (PUs) that can exist within an SNA network: PU.T5, PU.T4, PU.T2, and PU.T1. Units of type PU.T5 are host computers (370s). Communication controllers (e.g., 3704s or 3705s) are described as

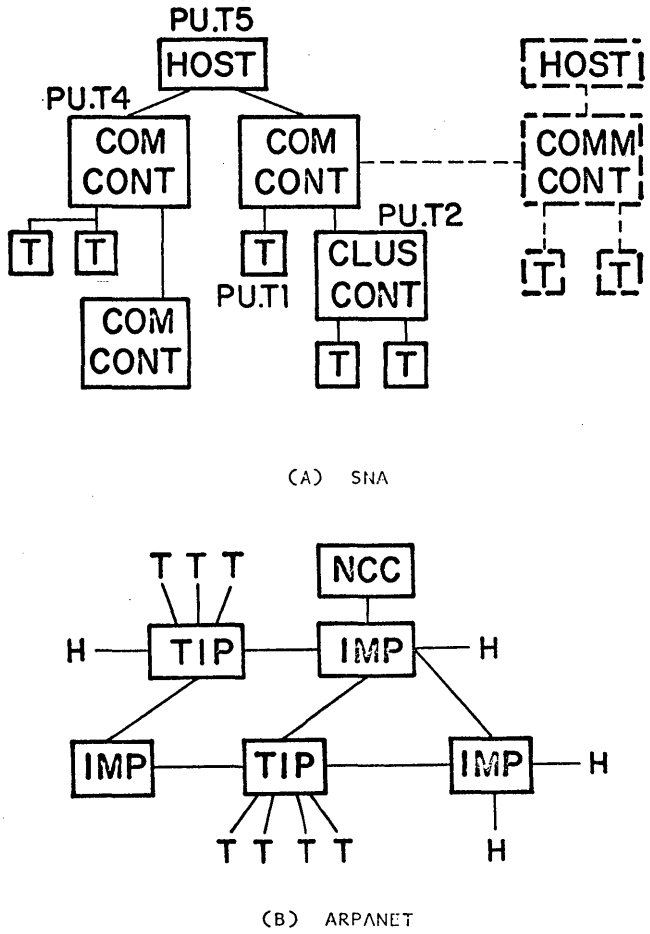


Figure 1—Physical network

units of type PU.T4. Cluster controllers (e.g., 3601s) are described as units of type PU.T2. Finally, terminals (e.g., 3270s) are units of type PU.T1. A basic concept of SNA is that there is only a single SNA node description and PU types are obtained as subsets of that single description. In this sense, PU.T1, PU.T2, and PU.T4 are subsets of PU.T5. The original announcement of SNA permitted configurations consisting of only a single host as shown by the solid lines. A more recent announcement from IBM has described expanded SNA implementations consisting of multi-host configurations (indicated by the dotted lines).

Figure 1b illustrates a typical physical network arrangement that may exist in the ARPANET. The two types of network nodes are referred to as IMPs and TIPs. The IMPs provide a store-and-forward switching function in addition to providing an interface for connecting host computers (H). The TIP is an IMP with an additional front-end concentrator function providing network access for unintelligent terminals (T). Arbitrary topological interconnections of IMPs and TIPs are permitted within the ARPANET design. A Network Control Center (NCC) provides a central facility for network monitoring, diagnosis and maintenance.

Logical network concept

Figure 2 contrasts the two network architectures with respect to their basic communication facilities. Within SNA the fundamental communication system is referred to as the transmission subsystem. The transmission subsystem provides process-to-process level communication between entities referred to as Network Addressable Units (NAUs). Three types of NAUs exist. The Logical Unit (LU) provides an interface port for SNA end-users. End users may be either human operators at a terminal or application programs. Physical Units (PUs) comprise a second set of network addressable units. A distinct PU is associated with each shared communications resource (host, cluster controller, etc.) in the network. The PU is the entitiy to which communication is directed when one wants to communicate with the associated physical device. Finally, each SNA network has one (more in a multi-host network) System Services Control Point (SSCP). The SSCP provides central monitoring, coordination, and control of its domain in the SNA network.

In the case of the ARPANET the IMP subnetwork consisting of interconnected IMPs (and TIPs) provides the basic communication facility. At this level the network hosts are the addressable units. It is important to point out that it is

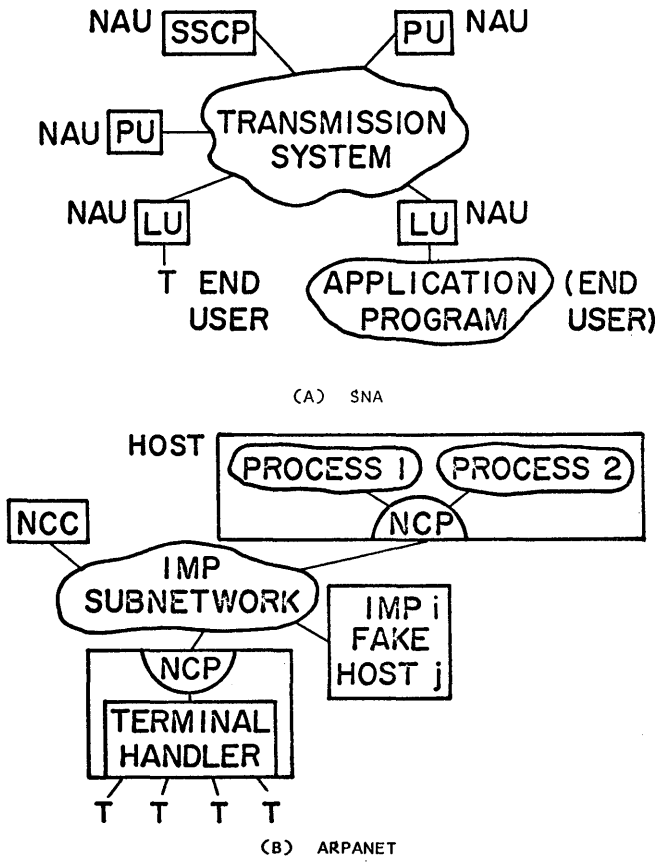


Figure 2—Logical network

the combination of the IMP subnetwork and the Network Control Program (NCP) software within the hosts which provides the process-to-process communication analogous to the SNA transmission subsystem.

Although there are not any logical entities within the ARPANET which correspond directly to the three types of NAU under SNA, certain analogies can be drawn between the two designs. Regular host computers (including the concentrator pseudo-host associated with each TIP) are the most common type of host subscribers interfaced to the IMP subnetwork. The internal processes supported by these regular hosts correspond to the logical units within SNA. Certain fake hosts such as PARAMETER CHANGE and DEBUG, on the other hand, exist within each of the IMPs and provide a means of effecting changes in network operation. They are, therefore, analogous to the physical units within the SNA design. Finally, the NCC facility can be viewed as analogous to the SSCP in the sense that it is the single network-wide coordination entity within the ARPA Network. A distinction between the SSCP and the NCC is the importance of the former for each conversation carried out over the network. In particular, it is directly involved in the initiation of LU-LU sessions as described in a later section.

Figure 3 illustrates the types of communication between addressable units under the two network concepts. A line in this figure indicates communication between the entities corresponding to its end points. We have assumed in the case of SNA that only a single SSCP exists. If multiple SSCPs exist, then these entities will clearly require communication between one another in order to synchronize their activities. In addition, in the case of the ARPANET, we have neglected end-user to fake host communication which, while possible, is only incidental to the primary communication capabilities supported by the ARPANET.

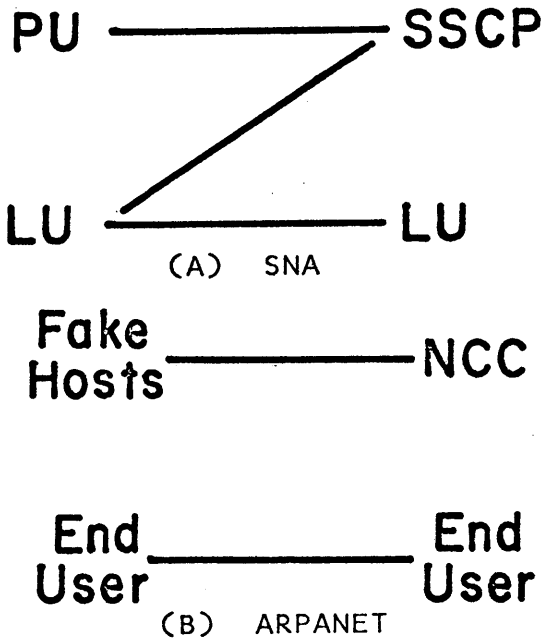


Figure 3—Types of communication supported

Key architectural characteristics

Figure 4 summarizes a number of selected architectural characteristics in the case of the two network designs. The first entry in the table indicates that both SNA and ARPANET are based on block or packet-switching technology. In the case of the ARPANET, a packet has a fixed maximum number of bits. In the case of SNA, the maximum packet size is not only implementation-dependent but can also vary from node to node within the network. Under both designs packets are passed through the network according to a routing algorithm. SNA has adopted a routing approach based on static tables within the nodes. The tables are centrally maintained by the SSCP. The ARPANET, on the other hand, has implemented a distributed adaptive routing algorithm in an attempt to dynamically adapt the flow of network traffic to changes in network topology and circuit loading. The routing algorithm is carried out periodically at each of the network nodes based on information that each network node receives from its neighbors describing the best path to each destination.

	SNA	ARPANET
Switching Technology	Store and Forward Packet Switching	Store and Forward Packet Switching
Routing	Centralized Static	Distributed Adaptive
Segmenting / Reassembly	Can be Performed at Each Node	Packetized at Source Node, Reassembled at Destination Node
Blocking / Deblocking	Yes	No
Sequencing	BIUs and BIU Segments Kept in Order Over Entire Path	Packets Reordered at Destination Node

Figure 4—Key architectural characteristics

The processes of segmenting and reassembly provide the means for translating between data units (SNA PIUs and ARPANET messages) presented to the network and packets switched through the network. Under SNA, segmenting and reassembly can be performed at each node along the source-to-destination path. Within the ARPANET, packetizing is done only at the source IMP and reassembly is performed at the destination IMP after all of the packets of the message have arrived. Blocking and deblocking are associated with the combining of logically unrelated packets into a single block (superpacket) for efficiency of transmission over in-

dividual network links. This concept exists only within SNA where it is used for efficiency of transmission over 370 channels. As indicated by the last entry in Figure 4, messages (BIUs) and message fragments are kept in order over their source-to-destination path within SNA, whereas ARPANET packets are routed independently and only reordered at the destination node. The ARPANET approach has potential benefits with respect to end-to-end delay and throughput but requires careful design of the control and buffering procedures in order to avoid the possibility of network lockups.[9]

The concepts of packetizing and blocking within the two network architectures are illustrated in more detail in Figure 5. This figure illustrates the data envelopes associated with transmitting an application buffer of data from one process to another via the network. In the case of SNA the application buffer is first split into a number of Basic Information Units (BIUs), each consisting of a Request Header (RH) and a Request Unit (RU). A set of related request units is referred to as a BIU chain. (Note: A BIU chain can also be assembled from multiple application buffers.) The BIU chain is treated as a unit with respect to acknowledgment and recovery procedures. The Request Headers contain only control information to support process-to-process level communication. Application data is contained within the Request Unit portion of the BIU. Before a Request Unit can be transmitted, a Transmission Header (TH) is attached as shown. The Transmission Header contains end-to-end control information such as destination address, sequence number, etc. The combination of Transmission Header, Request Header, and Request Unit is referred to as a Path Information Unit or PIU. If required, a BIU can be segmented into BIU segments (each of which then becomes a PIU), as shown, prior to transmission over any network link. The PIU with Data Link Control (DLC) appended at the beginning and end is ready for transmission over the network link and is referred to as a Basic Link Unit (BLU). As illustrated in the last two lines of Figure 5a, independent PIUs can be combined or blocked at a node as described previously for purposes of transmission efficiency.

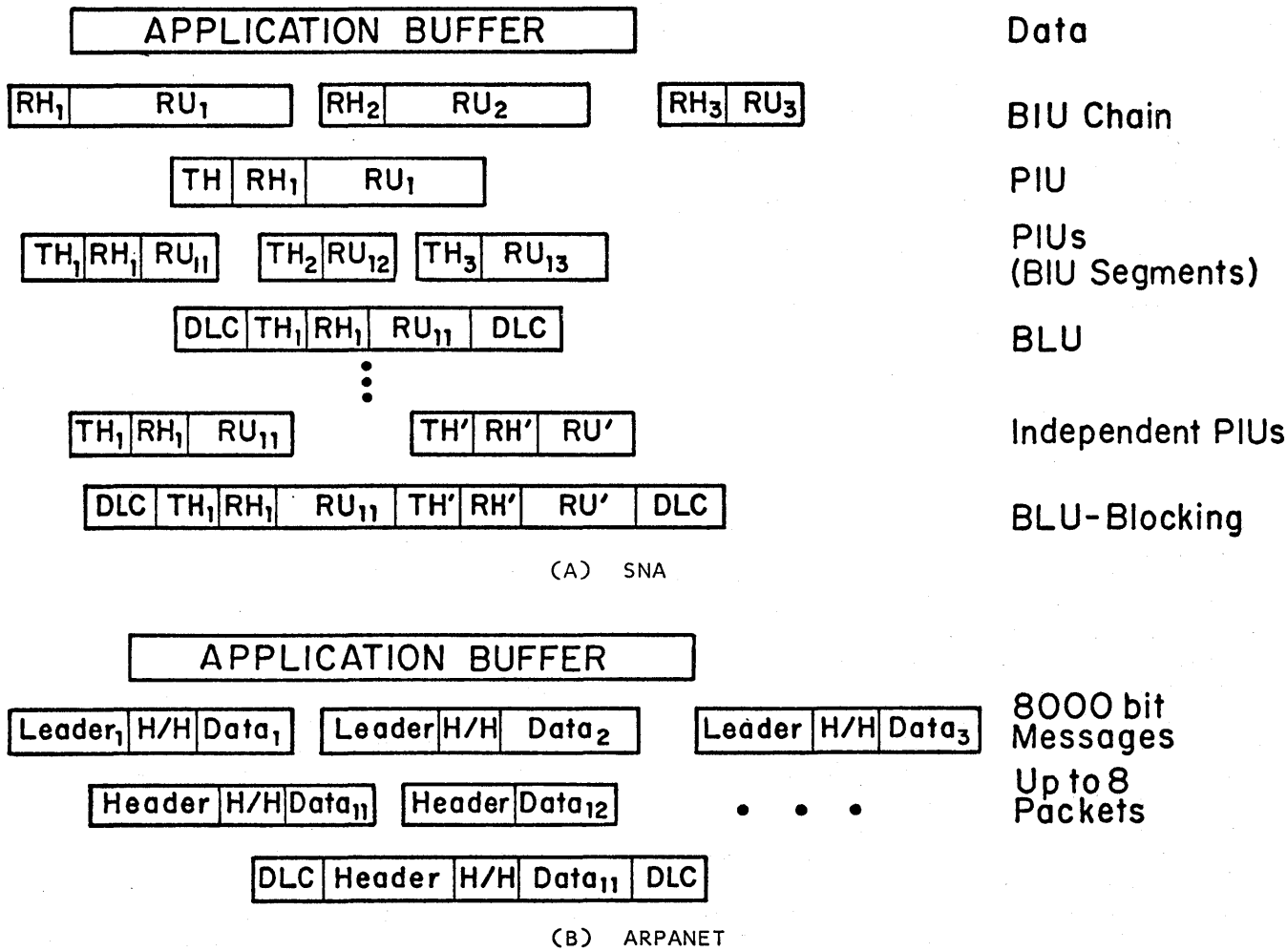


Figure 5—Data envelopes

The situation is somewhat simpler in the case of the ARPANET. Here an application buffer is split by the host into an arbitrary number of network messages, each message less than 8000 bits in length. Host-to-host protocol control information (H/H) is added and interpreted by the network control programs in each host in order to support process-to-process level communication. Each message is then divided into no more than eight packets of approximately 1000 bits in length. These packets are routed independently through the IMP subnetwork. Host-to-IMP control information is provided by a leader attached to each message transmitted by a host. This leader is used to create the control information contained in the header of each of the data packets.

Link control procedures

Two types of link control are supported under SNA, the 370 channel protocol and Synchronous Data Link Control (SDLC).[10] Analogous link control procedures supported in the ARPANET are the standard host-to-IMP interface,[11] the very distant host interface,[11] and the IMP-to-IMP protocol.[2] The most appropriate of these protocols for purposes of comparison are SDLC and the ARPANET IMP-to-IMP protocol.

Although SDLC and the ARPANET IMP-to-IMP protocol both support communication over transmission circuits, these two protocols differ along several dimensions. SDLC implements data transparency by bit insertion in the data stream. Five sequential one bits in a row are always followed by a zero in the data stream to distinguish such patterns from framing patterns (which involve a sequence of six sequential ones). Transparency in the ARPANET protocol is based on doubling of DLE characters. This procedure is similar to the one implemented under IBM's older link protocol, BSC, which was the basis of the IMP-to-IMP protocol frame structure.

Another difference between SDLC and the protocol used on the links between IMPs is the diversity of the data links supported. Under SDLC a variety of data link configurations are supported, including point-to-point, multi-point, loop, etc. The ARPANET IMP-to-IMP protocol, on the other hand, is only designed for dedicated full-duplex point-to-point links. A third distinction between the two protocols is in the fundamental relation between the two ends of the link. Under SDLC each link has a primary and a secondary end. The primary end is typically associated with the station having superior processing capabilities. Under the ARPANET protocol both stations are assumed to have identical capabilities and the two ends of the link are symmetric. A minor difference between the two protocols is the length of the CRC checksum added for error detection. This checksum is 16 bits in the case of SDLC and 24 bits in the case of the ARPANET.

Perhaps the most interesting contrast between the two link control procedures is illustrated in Figure 6. As indicated, under SDLC the link control procedure implements what is fundamentally a single logical channel capable of supporting eight outstanding (unacknowledged) frames. These frames are numbered sequentially in order to enable the transmitting end to associate acknowledgments with data units. The analogous arrangement between IMPs is a sequence of eight logical channels each of which can support only one outstanding frame. In a sense these alternate approaches are identical: if all frames are acknowledged properly, both approaches support the same throughput over the link. If a frame is not acknowledged, however, the SDLC procedure requires retransmission of everything following the unacknowledged frame, whereas the IMP-to-IMP protocol conserves channel bandwidth by requiring retransmission of only the unacknowledged data. A more fundamental difference is a result of the dependency between frames which for SDLC is inherent in the sequential numbering procedure. In particular, failing to acknowledge any single frame completely blocks the flow of data over the link within eight frames of the frame which was unacknowledged. In the case of the ARPANET protocol, on the other hand, failing to acknowledge an individual frame transmitted over any logical channel does not impact the transmission of new frames over the remaining seven logical channels.

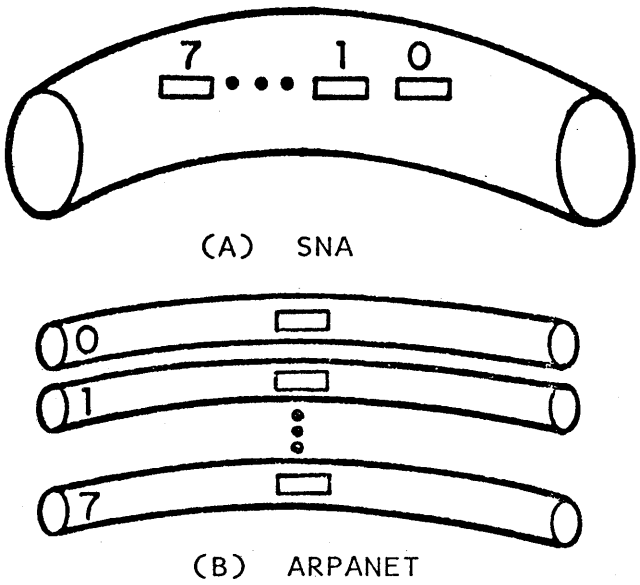


Figure 6—Link control

The ARPANET IMPs take advantage of this independence by permitting packet discard at the link control level. If a packet cannot be handled due to local congestion, the receiving IMP fails to acknowledge the packet as many times as necessary, being assured that its neighbor will continuously retransmit the data. Under SDLC this type of flow control cannot be carried out at the link level and an additional higher level mechanism must be incorporated for this purpose.

Error control

There are at least two types of errors which are associated with the transmission of data through packet switched networks. Each type of error is typically handled by a distinct mechanism. Corrupted data due to burst noise on the communications channels or errors within the communications hardware is detected by appending checksums to transmitted data. Lost, duplicated, and out of order data is typically detected by the addition of sequence number information to each frame transmitted. These mechanisms can be applied both over individual links and over end-to-end paths. Figure 7 illustrates the mechanisms for end-to-end error control which exist in the ARPANET and SNA designs. As shown in Figure 7a, there is a single end-to-end sequence number attached to each request unit transmitted by a process in an SNA network. This 16-bit sequence number is generated by the transmission control (TC) function of the source node. It is passed back to the source process in order that acknowledgment and error recovery information associated with that request unit can be identified. Errors due to corruption of the data are handled by checksums on the links (associated with SDLC).

In the ARPANET environment, sequencing is done on a host-to-host basis. Internal sequence numbers within the IMP subnetwork guarantee that messages are delivered to the destination host in the same order that they were submitted by the source host. A 12-bit sequence number ("message I.D.") field which is passed between the host and its IMP can be used for identification of responses (e.g., RFNMs) for multiple outstanding messages. It is also passed through to the destination host so that the destination can detect out of order errors due to subnetwork failures. As in the case of SNA, corruption of data on the lines is handled by link checksums. An additional end-to-end checksum is implemented in software in order to detect corruption of data originating within the nodes. This software checksum is computed on each packet by the IMP as it comes in from the source host and is checked at each node along the end-to-end path.

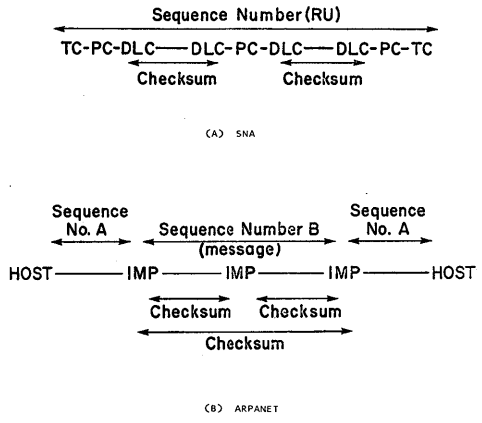


Figure 7—Error control

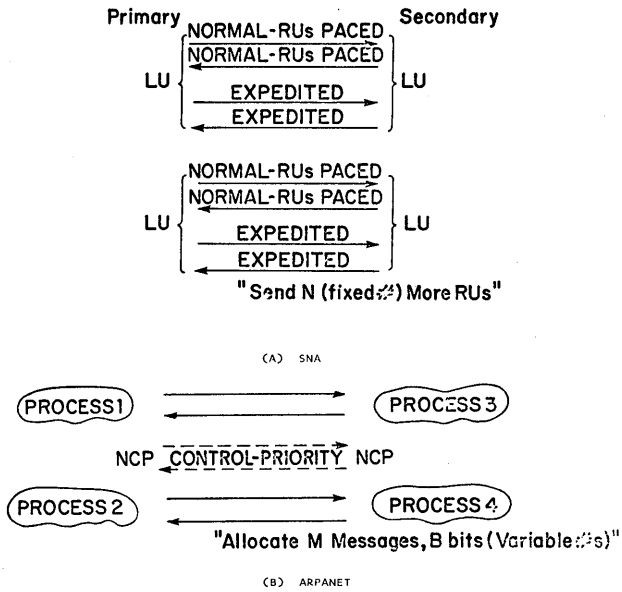


Figure 8—Flow control

Flow control

Figure 8 illustrates the type of process-to-process flow control implemented in the two network architectures. Under SNA, two logical units are always connected by two logical full-duplex flows, a normal flow and an expedited (priority) flow. The expedited flow is typically used for the transmission of process-to-process control messages. On the normal flow a form of flow control termed pacing is applied. Pacing messages sent from the receiver to the transmitter take the form of "Send N More RUs". Such pacing requests can be sent after fewer than N RUs have been received in order to keep the end-to-end logical channel full in support of high throughput applications.

In the ARPANET process-to-process communication typically is carried out over a single pair of simplex channels. Control messages between all processes on two communicating hosts share a common control connection over which messages are handled with priority. Flow control is applied in both directions and allocations take the form of "Allocate M More Messages and B More Bits." [12] Both M and B can vary from one allocate message to the next.

Both of these flow control schemes can be described as incremental. This means that they both take the form of the receiver telling the transmitter that a certain number "more" data units are allowed. It has been argued[13] that such incremental flow control schemes have the potential for loss of synchrony if allocated messages are lost within the communications system and no notification of this loss is provided. Windowing schemes tied more tightly to end-to-end sequence numbers have been proposed recently.[14,15] Such windowing schemes have the property that they are self-synchronizing.

Addressing

A major difference between process-to-process addressing in the SNA and ARPANET architectures is the use of network names for NAUs under SNA. Names or logical addresses have the advantage of facilitating transparent movement of services between physical facilities by separating the concept of network name from the location of the named facility. Mapping between names and addresses under SNA is handled by the System Services Control Point during session initiation as described in the next section. Under the ARPANET design end-user identification is tied to physical location.

The formats of the network addresses themselves provide another interesting contrast. SNA network addresses consist of 16 bits. The 16 bits are split between a sub-area address (used for store-and-forward routing) and an element within the sub-area (used for local routing). ARPANET network addresses consist of a 16-bit IMP number (used for store-and-forward routing), an 8-bit host number (used for local routing), and a 32-bit socket number (used for process identification). In order to avoid having to transmit a pair of 32-bit socket identifications with each message transmitted between processes, an 8-bit shorthand called the "link I.D." is used in the ARPANET. This puts a (realistic) limit on the number of simultaneous conversations between any pair of hosts.

Session initiation

Figure 9 illustrates the procedures involved in setting up a logical session (connection) between two end users under the SNA and ARPANET architectures. The key point to note in this figure is that session initiation under SNA involves the SSCP as well as the two communicating processes, whereas session initiation within the ARPANET involves only the two communicating processes themselves (and their NCPs). Figure 9a illustrates the sequence of exchanges assuming a primary LU wishes to establish a connection to a secondary LU. The primary LU first sends an INITIATE command to the SSCP. The INITIATE indicates the type of session requested, the name of the LU with which the primary wishes to speak, and password information for authenticating the request from the primary. If the establishment of the requested connection is permitted by the SSCP, it will respond to the primary by the transmission of a CONTROL INITIATE command. The CONTROL INITIATE command contains a "bind image." The bind image includes all of the parameters which are subsequently sent in a BIND command from the primary to the secondary LU. The parameters in the BIND command specify the type of session being established in detail. Assuming that the secondary accepts the BIND request, the primary notifies the SSCP via a SESSION STARTED message. Session authorization checks carried out by the SSCP are associated with the allocation of reusable resources as well as the name-to-address map.

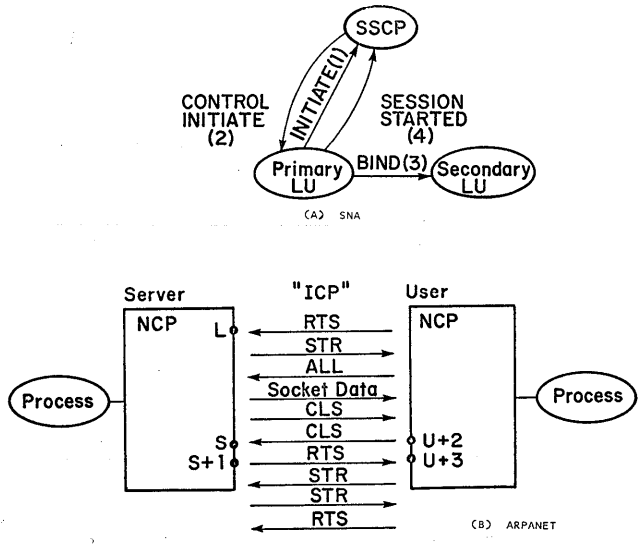


Figure 9—Session initiation

Communication between a user and a server process in the case of the ARPANET is established in one of two ways. For a service process which handles only one user process at a time, a simple pair of connection requests is exchanged. For multi-user server processes supporting multiple simultaneous conversations, a standard procedure referred to as the Initial Connection Protocol (ICP)[16] is implemented for session initiation. The set of control messages which comprise the ICP is illustrated in Figure 9b. The complexity of the required exchange is a result of the fact that connections are fundamentally simplex under the current ARPANET protocol standards (two are typically used for process-to-process communication), and the fact that two connections cannot have either of their ends in common. The ICP works by having the server process continually listening for connections from users on a "well-known socket" (L in the figure). A user wishing to obtain the service connects to this well-known socket and is subsequently switched by the server from the well-known socket to another pair of sockets which the server chooses. The server then continues to listen on socket L for connections from other remote users. In contrast to the tailoring of process-to-process communication which under SNA is carried out as the result of the BIND exchange, tailoring of the session between user and server processes is a higher level protocol function. The ARPANET supports only a single type of process-to-process communication.

Word length mismatch

The issue of word length mismatch was of great concern to the designers of the ARPANET. This is due to the fact that the IMPs are 16-bit machines, and the hosts supported are heterogeneous. Attached host systems include IBM 370s (32-bit word length), PDP-10s (36-bit word length), PDP-1s

(18-bit word length), minicomputers (16-bit word length), and CDC6600s (60-bit word length). Word length mismatch is not so important an issue under SNA because equipment from only a single vendor is involved. In addition, most IBM mainframes and peripherals have byte handling facilities.

In order to neutralize the problems of word length mismatch, the ARPANET designers provided two capabilities. The first enables the transmission of messages containing arbitrary numbers of bits (less than the maximum message size) through the network. The second facilitates the combining of multiple successive messages into a single application buffer without excessive bit shifting. The first goal was achieved by having the host/IMP interfaces add and interpret hardware padding sequences. The IMP interface at the source adds padding bits at the end of each message it receives from its host. The padding consists of a 1 followed by as many zeros as are required to fill out an IMP word. The host interface at the destination adds as many additional zero padding bits to the message as are required to fill out the word length of the receiving host. To avoid excessive bit shifting, the host-to-host protocol is defined to have a "message header." The length of this header in conjunction with the leader associated with each message insures that the text portion of the message starts in a "good place" for most machine word lengths (i.e., on a word boundary). In addition, the host-to-host protocol specifies a "connection byte size" which can be used to guarantee that the message text also ends in a "good place."

SUMMARY AND CONCLUSION

The previous sections have detailed and contrasted the mechanisms supporting a variety of network functions in the ARPANET and in SNA. An important similarity of approach is the hierarchical layering of functionality and protocols which is characterisitic of both designs. Communication in both cases is characteristic of both designs. Comnication in both cases is carried out between pairs of peer functional layers in the two communicating entities. This approach has advantages with respect to ease of comprehension, maintenance, debugging, and expansion. Both designs are fundamentally packet switching technology and both designs involve some amount of physical separation as well as logical separation of data communication and data processing.

More interesting than the conceptual similarities are the key architectural differences. These differences are more evident in SNA implementations than in the definition of the architecture itself. Centralization of network functionality is one such critical difference. Perhaps due to its ancestry, centralization is more clearly in evidence in the SNA design. A key example of this is the SSCP involvement in session initiation in single SSCP implementations. Symmetry of peer elements with respect to control is a second important distinction. Two communicating entities in an SNA environment often communicate as a primary to a secondary. One logical (physical) end of the communication path (link) is endowed with more control capability than the other. In the ARPANET, on the other hand, communication is typically symmetric (primary to primary). A third important distinction between the two designs is the nature of the routing strategy. The distributed adaptive approach developed for the ARPANET is one of the key features of that design. While such routing strategies are not necessarily excluded from implementation under SNA, there does not appear to be a strong leaning in this direction on the part of the SNA designers. Finally, we point to general complexity as an area where the two architectures differ. The architecture of SNA is capable of supporting both half and full duplex links, switched and non-switched links, multipoint links, and loop configurations. SNA session protocols are tailored at the transmission subsystem level for efficiency in contrast to the single process level communication protocol within the ARPANET. In addition, a variety of network elements with different physical capabilities are accounted for under SNA, whereas only a single type of network element exists in the ARPANET design.

The previous sections have attempted to present a side-by-side comparison of the ARPANET and SNA architectures. The goal of this comparison was not only to provide useful information but also to raise questions and spark some interest in comparing the relative merits of different architectural approaches. Moreover, by carefully examining two points in the space of possible architectural implementations, it is possible to gain a better understanding of additional architectural alternatives.

ACKNOWLEDGMENTS

The author would like to thank John Davidson, Alex McKenzie, Ira Richer and Eric Rosen of BBN and James P. Gray of IBM for their critical review of this paper, and Jan Mahoney and Julie Moore for their assistance in its preparation.

REFERENCES

1. Roberts, L. G. and B. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings of the Spring Joint Computer Conference,* Atlantic City, May 1970, pp. 533-549.
2. McQuillan, J. M. and D. C. Walden, "ARPANET Design Decisions," *Computer Networks,* Vol. 1, No. 5, September 1977.
3. Walden, D. C., "Host-to-Host Protocols," in *Network Systems and Software, Infotech State-of-the-Art Report 24,* Infotech Information Ltd., Nicholson House, Maidenhead, Berkshire, England, pp. 287-316.
4. IBM Corporation, "System Network Architecture General Information," Publication GA27-3102-0.
5. IBM Corporation, "System Network Architecture Format and Protocol Reference Manual: Architectural Logic," Publication SC30-3112-0.
6. Piatowski, T. F., P. C. Hull, and R. J. Sundstrom, "Inside IBM's System Network Architecture," *Data Communications,* February 1977, pp. 33-48.
7. Gray, J. P., "Network Services in Systems Network Architecture," *IEEE Transactions on Communications,* Vol. COM-25, No. 1, January 1977, pp. 104-116.

8. IBM Corporation, "Introduction to Advanced Communications Function," Publication GC30-3033.
9. Kleinrock, L, *Queuing Systems: Volume 2—Computer Applications,* Wiley Interscience, New York, 1976, pp. 438-458.
10. IBM Corporation, "Synchronous Data Link Control General Information," Publication GA27-3102.
11. Bolt Beranek and Newman Inc., "Specification for the Interconnection of a Host and an IMP," Report No. 1822, January, 1976.
12. McKenzie, A., "Host/Host Protocol for the ARPA Network," in *ARPANET Protocol Handbook,* Network Information Center, Stanford Research Institute, Menlo Park, California, April 1976, pp. 7-37.
13. Cerf, V., "An Assessment of ARPANET Protocols," RFC 635, NIC 30489, April 1974.
14. Cerf, V. and R. E. Kahn, "A Protocol for Packet Network Interconnection," *IEEE Transactions on Communications,* Vol. COM-22, No. 5, May 1974, pp. 637-648.
15. Cerf, V. G., A. McKenzie, R. Scantlebury, and H. Zimmermann, "Proposal for an International End-to-End Protocol," *ACM Computer Communications Review,* Vol. 6, No. 1, January 1976.
16. Postel, J., "Official Initial Connection Protocol," in *ARPANET Protocol Handbook,* Network Information Center, Stanford Research Institute, Menlo Park, California, April 1976, pp. 41-49.

Design of a message processing system for a multilevel secure environment*

by STANELY R. AMES, JR.
The MITRE Corporation
Bedford, Massachusetts

and

DONALD R. OESTREICHER
USC Information Science Institute
Marina Del Rey, California

INTRODUCTION

A primary requirement for a message system in an operational military environment is that it be secure enough to process messages at multiple levels of classification. But for the system to be accepted, the operator interface must be "usable." Usability relates to three things: features provided, ease of entering commands, and overall performance. Certainly an interface that does not perform well in these respects, or is difficult to learn or use, will not be used.

The Department of Defense Advanced Research Projects Agency (DARPA) and the Navy are sponsors of an experiment to evaluate the operational use of a computer-aided message handling system at PACOM Headquarters in Hawaii. The experiment will evaluate the operational and organizational impact of the automated service on a community that now uses a largely manual system. The purpose of this paper is to document the security design of SIGMA,** the system that will be used in the experiment.

In the following section, a description of the SIGMA message processing system is given. The third section provides background and discusses the kernel approach to multilevel security. In the fourth section we describe several security problems encountered in the design. The fifth section presents the design of the SIGMA message service. The additional features that the kernel must provide to support SIGMA efficiently are documented in the sixth section. Finally, a summary is provided to highlight the paper's main points.

* This research was supported by the Defense Advanced Research Projects Agency under ARPA Order Nos. 3338, 2223 Contract Nos. F19628-78-C-0001, DAHC15-72-C-0308, MITRE Project No. 8010, ISI Project Nos. 3D30 and 3P10. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency or the United States Government.

** The SIGMA message service is one of three services developed during this experiment. The other two are the HERMES system built by Bolt, Beranek, and Newman, Inc.[1] and DMS built by Massachusetts Institute of Technology [2].

THE SIGMA MESSAGE PROCESSING SYSTEM

Information Science Institute of the University of Southern California developed SIGMA specifically to meet the message handling needs of a military command. SIGMA is a secure interactive message handling system providing computer-aided message handling services for the receipt, filing, retrieval, creation, and coordination of military (AUTODIN) messages. We consider it secure in that it presents an interface to the user constrained to abide by the DoD security policy. It is an interactive system since all user-system communications occur via an on-line terminal with a CRT display. Finally, it is a message handling system because it supports the typical message processing functions needed by any formal organizational operation.

SIGMA supports the full cycle for processing incoming and outgoing messages in a military operation. It provides flexible filing capabilities for on-line storage of all messages. Easy access to messages and files is provided by selective search and retrieval functions. Incoming AUTODIN messages move through the system by informal forwarding or by formal action assignment. Outgoing messages are processed by a set of functions supporting message creation, editing, coordination, release, and post-transmission comeback copies.

SIGMA operates on a DEC PDP-10 computer with the TENEX operating system. AUTODIN messages enter through the local AUTODIN message exchange. SIGMA distributes these messages to all pertinent addressees on the system where each user can access them through his SIGMA display terminal.

THE KERNEL APPROACH TO MULTILEVEL SECURITY

The need to process multiple levels of classified data led the Air Force Electronic Systems Division to sponsor several research and development efforts to build an operating

system that could satisfy by technical verification that DoD security requirements had been met.[3] Many of the results of the ESD work have been borrowed in the design of SIGMA, specifically adherence to a mathematical model based on the concept of a reference monitor—an abstract mechanism that controls the flow of information within a computer system by mediating every attempt by a subject (active system element) to access an object (information container).*[4] The hardware/software mechanism that implements the reference monitor is called a security kernel. The kernel uses the rules of the mathematical model[5,6] as a specific policy for mediating access requests. This incorporation of policy into the kernel allows for a proof that verifies that the kernel correctly applies the policy to the information it protects.[7,8]

The mathematical model establishes an "inductive nature" of security by demonstrating that security is preserved from one state to another. Security is defined with two rules: the simple security condition and the *-property.[9] The former states that a subject (active entity) cannot observe the contents of an object (information container) unless its security level is greater than or equal to the security level of the object.** The latter further restricts possible access by stipulating that a subject may only modify an object if that object's security level is greater than or equal to the security level of the subject.

The purpose of the simple security condition is to prohibit users from obtaining data that they are not entitled to see. The *-property is designed to prohibit a program operating on behalf of a user from reducing the classification of any information.

When a user is given a clearance, he is charged with responsibility for maintaining the classification of classified information. A computer utility cannot necessarily be given this same trust. This is due to: the amount of information that may be compromised; the speed with which the compromise may occur; and the difficulty in detecting or apprehending the violating program. By enforcing the *-property on computer programs, a program will not be able to either accidently or maliciously compromise information. Designers of computer utilities constrained by the *-property must ensure that *-property enforcement does not unnecessarily restrict the capabilities of the user.

The enforcement of the *-property allows us to reduce the volume of code that needs to be trusted to a central section of the operating system. This central section of the operating system is the software component of the security kernel. To provide security, a kernel must (1) mediate every access by a subject to an object, (2) be protected from unauthorized modification, and (3) correctly perform its functions. A kernel satisfies the first requirement by creating an environment within which all non-kernel software is constrained to operate and by maintaining control over it.

The requirement to protect against unauthorized modification is satisfied by isolating the security kernel software in one or more protection domains, for example, by a ring mechanism.[10] Finally, the requirement that the kernel correctly perform its functions is satisfied by using a formal methodology. A suitable methodology was introduced by Bell and Burke.[7] It includes: (1) a proof that the kernel behavior enforces the desired policy;[11] and (2) a proof that the kernel is correctly implemented with respect to the description of its behavior used in the first step.[12]

We designed SIGMA with security kernel technology in mind. However, due to the absence of a kernel on the PDP-10 (the machine we used), the current implementation was done without a kernel. We have rigorously scrutinized the SIGMA design to ensure that the user interface provided would remain unchanged should SIGMA be reimplemented on a security kernel. In addition, the security primitives have been evaluated to ensure that their usefulness warrants their being included in a kernel.

SECURITY PROBLEMS OF MESSAGE PROCESSING SYSTEMS

A kernel supporting the current mathematical model of the DoD security policy is well suited for certain environments, such as a programming environment in which users operate at a single security level for long periods of time.[13] A message processing environment presents several problems not found in previous environments, including (1) the dynamic nature of the user's "working security level"; (2) the desire to present to the user information at more than a single security level; (3) the desire to accurately inform the user of the security level of all information he is reading and writing; and (4) the ability of users to extract text information and place it in a message of a lower classification than the source.

The user's "working security level" in a message system environment is considerably more dynamic than in the programming environment. Each time that a user performs an action on a different message, his working security level may have to change; for example, a user reading a Secret message may generate an Unclassified reply. While we could require the user to process messages at a single security level at a time, the resulting user interface would be clearly unacceptable to the user.[14]

To deal with these problems a new approach is needed that includes: a terminal that will allow users to process information at more than one security level at a time; and trusted processes that are able to violate the security rules in a controlled manner. The next section describes the security architecture of the SIGMA message processing system.

SECURE MESSAGE PROCESSING SYSTEM ARCHITECTURE

The SIGMA security design has two goals: to produce a certifiably secure service, and to present the user with an agreeable user interface. In many situations these goals are at cross-purposes. Our general approach has been to present the user with a true picture of what is happening, maintain

* In a computer system, subjects are users and processes, and objects include programs, data files, and peripheral devices.

** Currently the security levels used in SIGMA are only the four standard DoD classifications, i.e., Unclassified, Confidential, Secret, and Top Secret.

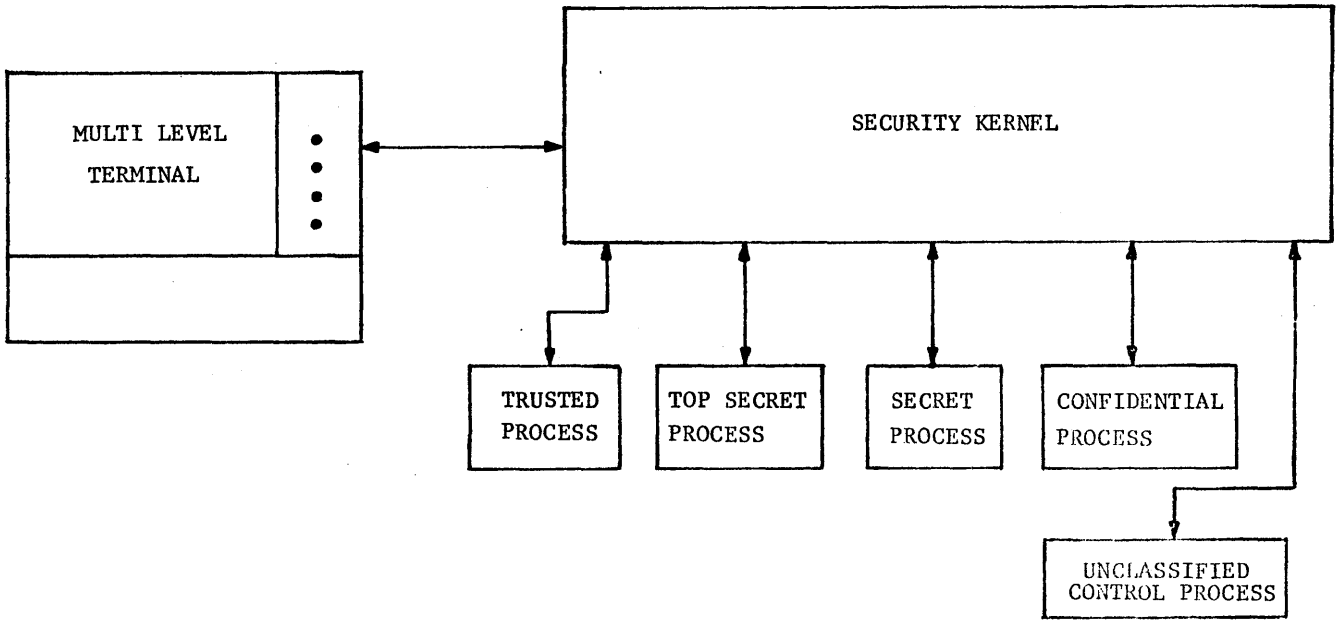


Figure 1—Sigma process structure

the user's data at the proper level (or higher if this is not possible), and make it convenient for the user to do the right thing.[14]

Overview of the design

When using SIGMA, a user is actually interacting with a collection of up to five processes (see Figure 1). These are the trusted process, an unclassified control process, and one process for each classified level that the user/terminal is cleared to operate. Each process (except the trusted process) can write data only at its own level and can read data at its level or lower.

SIGMA attempts to be as helpful to the user as it can, organizing the user's session and cleaning up the user's state (current context) as necessary. The service also attempts to understand the user's current context and conform its behavior to the situation. For this reason the context information must be available to all user processes; thus, it must be unclassified.

The user's state in SIGMA is divided into two parts. The first part contains the current list of objects being accessed and functions being performed by the user. This portion of the state is maintained at the unclassified level by the unclassified control process. The second part contains the current list of message entries (from the open message file) in which the user has expressed an interest. The entry list information is potentially classified at the level of the file and is thus maintained at this classification level by the appropriate classified process.

This dichotomy of state is reflected directly into the security design. Commands are divided into those which access the unclassified state (unclassified commands) and those which access the entry list (classified commands). The latter group includes both commands that use the entry list for input and those that allow the user to enter classified information as part of the command.

Multilevel terminal

We designed the terminal, used by SIGMA, to enable the user to interact with data at more than one security level at a time. The screen of this "multilevel terminal" is divided into "windows" (see Figure 2), each of which is logically an independent terminal. Each window scrolls independently and may have a different security level. Windows are further divided into domains that have various attributes (e.g., enterable, editable, underlined, etc.). The domain's security level is the same as that of the window.

To keep the user apprised of the level of information he is viewing and entering, we added two sets of lights to the terminal. Each set consists of four lights (one for each security level); one and only one light of each set is on at any time. The first set is mounted on the keyboard; it specifies the classification of the window in which the cursor currently resides. If the user wishes to know the level of any particular piece of information on the screen, he may move his cursor to the information. The second set of lights is mounted next to the screen and specifies the maximum level of information displayed on the screen.

Form of command input

We designed command input so that it could be done through a separate window that is normally at the unclassified level in order to keep the majority of the user's state information at unclassified. Certain commands, such as the

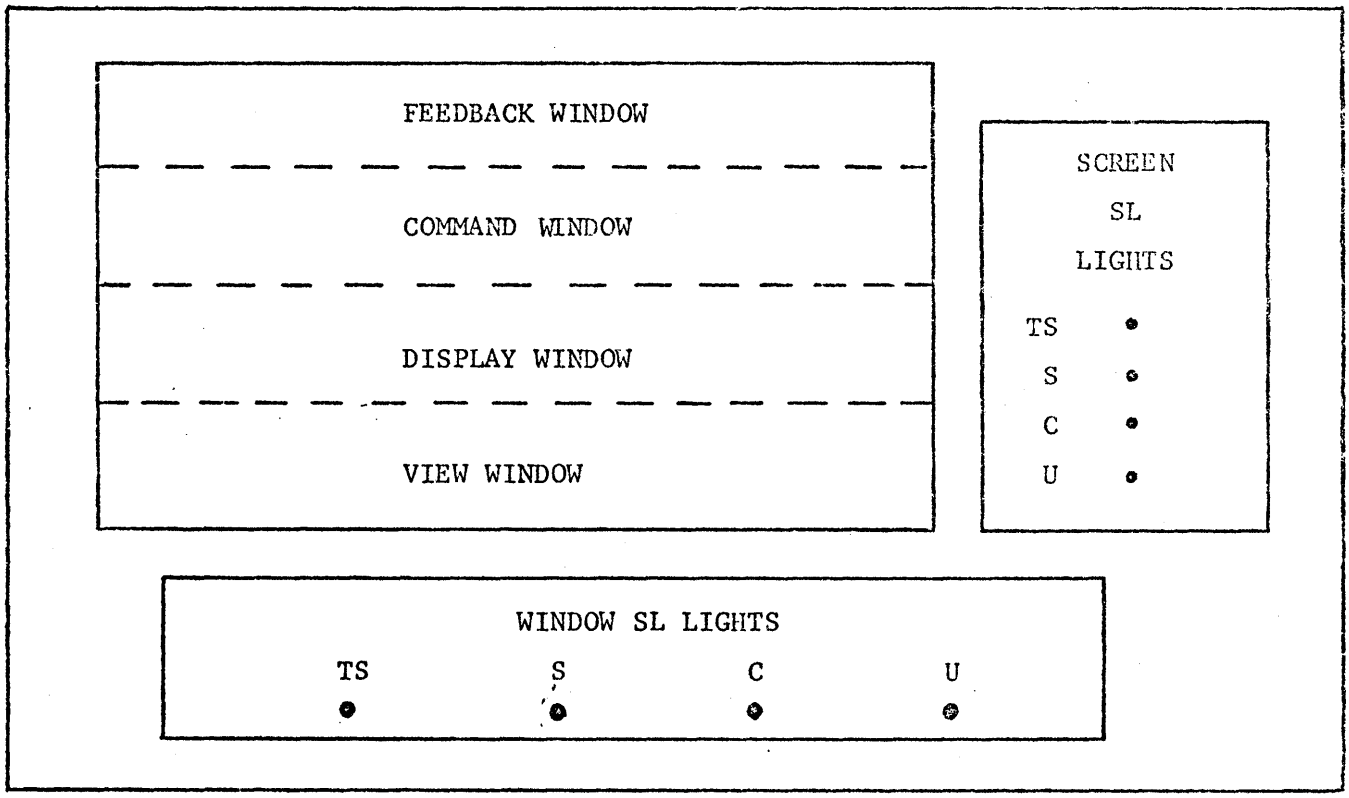


Figure 2—Multi level terminal

"find text string" command, have potentially classified arguments. For these commands the security level of the command window is raised to the level of the object that the command is affecting before the user enters the parameters.

Strict enforcement of the security model eliminates any possibility of a security compromise: a write-down path through the system that could be used to release information of a higher security level to a lower security level.[15] However, even with the enforcement of the model, there are several situations in a message system where the user, by following instructions given by the system, can inadvertently compromise small amounts of information.

Consider the following example: A user asks for a list of all his messages with a subject having word "x" in them. To perform this operation, the user must be at the security level of the file that he is looking at—greater than or equal to the security level of all the messages within that file. The enforcement of the *-property forces the result of this examination to be at the level at which the examination was performed—the security level of the file. Should the user then decide to perform any modification to a message returned by this examination that has a security level lower than the file security level, the *-property would require him to: issue commands at the security level of the message that he desired to modify, and tell the system the unique identification of the message told to him by the classified process. (The unique identification is required here because the system is unable to pass the desired identifier "down" due to the enforcement of the *-property.) However, this transmission through the user of the message-id from the higher process to the lower process is, itself, a violation of the *–property. Although it is conceivable that a maliciously written program could use this *-property violation to compromise information, we assume that the user serves as an effective filter in this write-down path (both in "bandwidth" and in checking for reasonableness), thereby precluding any reasonable software means of making use of this path.

Because of the hardships that strict enforcement of the *-property imposes on the user and because of the existence of *-property violations, a case can be made to ease the user interface in situations where this type of violation exists. The improvement takes the form of allowing SIGMA, in violation of the *-property rule of the security model, but with user concurrence, to write-down the unique identifier of the message that the user wants to modify. We limit the bandwidth of this type of *-property violation, so that it is no larger than the path that otherwise occurs through actions of the user, by allowing only a specific amount of fixed-format information to be transmitted to a lower security level and then only if the user has depressed an appropriate function key that is linked directly to the security kernel. Allowing the system to transmit this information greatly simplifies the user interface.

Trusted process functions

Certain functions need special capabilities to operate (such as the passing of message identifiers) but are relatively message-system dependent and thus are not included in the security kernel. We group these functions together in a "trusted process" that has the ability to transfer information in a controlled fashion in violation of the *-property.

The trusted process in SIGMA performs four functions: change classification; message release; command completion signals; and entry list transmission.

Change classification

SIGMA allows users to change the classification of text that they are allowed to access. When this happens, the trusted process clears the screen and presents the text in a simple fashion (19 lines at a time) for confirmation. When the user has confirmed the entire object, the trusted process logs this action and passes the text to a process at the new security level for refiling.

Message release

In the military formal messages are released with the commander's signature. Therefore, we consider the act of releasing a message a security event. To control message release, we require that the trusted process insure that the user requesting the release is authorized to release and that this user is making the request from an authorized terminal.

An additional security consideration with message release is that some AUTODIN terminals (ours in particular) treat the message header as unclassified. In SIGMA this header is created in the same window as the message text. Therefore, releasing a message implicitly lowers the classification of the header information. During message release the trusted process requires the user to confirm that all of the header information is actually unclassified. The trusted process logs this action before releasing the message.

Command completion signals

We have based the SIGMA design on the concept of an unclassified process that receives the majority of the commands, determines the proper security level needed to execute these commands, and then activates a process at that security level to perform the execution. The disadvantage of this approach is that, should an error occur between the unclassified control process and the classified operational process, the classified process cannot ask for clarification. Thus error recovery is difficult. This problem is referred to as the open loop problem.

Presently we believe that the best solution to the open loop problem is to allow the trusted process to close the loop when an error of this type is encountered, provided the user has depressed a function key since the last such request. Closing the loop improves recovery but has an impact on security, since a *-property violation exists when the loop is closed. As in command input, requiring a user action between successive writedowns restricts the bandwidth of this operation.

Entry lists

When a user process needs to write down a list of message identifiers, "entry lists," it passes this list to the trusted process for user confirmation directly. The trusted process checks the format and bounds of the entry numbers, and asks the user to directly confirm the number of entries being processed at each security level. Thus, the user has the ability to directly monitor (and control) the bandwidth of the writedown channel. This separate step is reasonable even from the user interface side, for if the number is too high or too low, the user can see that he specified his request incorrectly.*

KERNEL REQUIREMENTS

In order to be able to support the SIGMA architecture, the security kernel must provide certain additional features not found in kernels designed to date, including a terminal multiplexor for the multilevel terminal, a variety of object sizes, the ability to support large numbers of processes, an efficient inter-process communication facility, and a policy that can support "trusted" processes.

Multilevel terminal certification

Since the terminal supports the simultaneous display and editing of data at different classifications, we must demonstrate that the terminal (1) maintains the proper levels for all information it contains (possibly 20,000 characters) and (2) marks all information returned to the computer with the proper security level. It is the terminal's responsibility to assure that no information entered in a window by either the user (doing local editing) or the application computer is transferred to any other window. While at first pass the certification of the terminal may seem trivial, one must consider that the terminal code is currently produced for a single INTEL 8080 and occupies 32K bytes of PROM. Eventually multiple 8080's, application of Denning's flow analysis,[16] or the introduction of a kernel in the terminal will be necessary to guarantee separation of the windows.

Terminal multiplexor

A significant problem is the method for attaching the multilevel terminal to a secure system. We have identified two

* If the entry list has only one element, then the appropriate function key is sufficient and the further "confirm 1 entry" step is omitted. This operation allows the user to point to a single entry or mention it by number or context (CURRENT, NEXT) for a display, reply, file, etc., without being required to do more than use the proper function key to enter or confirm the command.

alternatives: each window could have a unique connection to the system or the kernel could multiplex all information to a terminal over the same communication line. We have chosen the multiplexing approach in order to minimize the number of terminal lines.

Communication between the system and the terminal is in the form of NOTICES and DISPATCHES.* The terminal multiplexor must assure that each NOTICE received from the terminal is directed to a user process whose security level is the same as the security level of the window. The multiplexor must also forward Function Key Notices to the trusted process to provide for the capability for a controlled write-down of message identifiers.

The terminal multiplexor must insure the correctness of all DISPATCHES to the terminal. With the exception of "window allocation" DISPATCHES, the terminal multiplexor need only check the window identifier and length to assure that the user process is communicating with a window to which it has access. All requests for terminal window allocations and deallocations must be done by the unclassified user control process. This process provides the terminal multiplexor with the security level for all newly created windows. The unclassified process can, at a later time, request a change in a window classification by notifying the multiplexor. If this new security level is lower than the current window security level, the multiplexor must erase the information currently in the window.

Process structure

The design of the kernel's process structure will have significant implications for the performance of SIGMA. On traditional timesharing systems, such as TENEX, process creation is expensive, and process swapping is lengthy. In order for SIGMA to operate efficiently the kernel must be able to (1) support large numbers of processes; (2) allow for fast process creation and deletion; and (3) swap processes with little overhead. Large numbers of processes are required because SIGMA requires several active processes per user. If SIGMA is extended to handle compartmented intelligence, fast process creation and deletion would be required. Finally, because large numbers of processes are doing small amounts of processing, process swapping occurs often. To expect a kernel to provide this type of support may require significant hardware support.

Interprocess communication

Equally important to the efficient operation of SIGMA are the speed and types of interprocess communication provided for by the kernel. SIGMA will require the kernel to support both preemptive (interrupt-like) and non-preemptive (message-like) types of interprocess communication. In addition, the latter mechanisms must support small messages for passing message-ids and large messages for transmitting entire command strings.

File system

The file system is often one of the most complex portions of the kernel, a quality which can cause unnecessary overhead. For example, SIGMA does not require the kernel to provide a file organization such as a directory hierarchy. A "flat file" system is entirely adequate and can be used more efficiently. The only special requirements for SIGMA are that the kernel should support both small files (512 bytes) and large files (10K bytes).

System integrity controls

To support SIGMA the security kernel must provide a mechanism that implements the notion of least privilege. This mechanism has been given the name "System Integrity."[17] SIGMA uses three separate privileges: a secure write-down privilege used by the trusted process to reclassify text; a release privilege used to restrict the releasing of messages to a select group; and a system security officer privilege used to initiate and set the security level of new users.

The primary rule that the system integrity controls must obey is: to modify an object or execute a kernel call a subject's system integrity level must be greater than or equal to the system integrity level of the object or kernel call.[18] There are no rules on reading or executing programs (programs in execution use the system integrity level of the process that they are executing under). We must therefore demonstrate that each subsystem with a system integrity level greater than system low (non-kernel, no privileges) does not execute any programs other than ones that we know execute properly.

CONCLUSION

The design of SIGMA demonstrates that it is possible to build a secure message processing system based on the kernel approach using the DoD security model. We have shown the refinements to the security policy that are needed to achieve a usable interface and have documented the features that a security kernel must provide to support a secure message processing system efficiently. The techniques used in designing SIGMA should be directly applicable to other transaction-oriented or data base management systems.

REFERENCES

1. Bolt, Beranek, and Newman Inc., "HERMES Message System Information Package: MME-HERMES an Introduction," draft in preparation, Bolt, Beranek, and Newman Inc., February 1977.
2. Vezza, A. and M. S. Broos, "An Electronic Message System: Where Does It Fit?" *Trends and Applications 1976: Computer Networks*, Gaithersburg, Maryland, November 1976, pp. 89, 97.

* NOTICES and DISPATCHES are packets of information to and from the host computer respectively.

3. "ESD 1974 Computer Security Developments Summary," MCI-75-1, Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, December 1974.
4. Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I and II, James P. Anderson & Co., Fort Washington, Pennsylvania, October 1972.
5. Bell, D. E. and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," M74-224, The MITRE Corporation, Bedford, Massachusetts, October 1974.
6. Walter, K. G., W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, Jr. and D. G. Shumway, "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.
7. Bell, D. E. and E. L. Burke, "A Software Validation Technique for Certification, Part 1: The Methodology," ESD-TR-75-54, Volume I, The MITRE Corporation, Bedford, Massachusetts, April 1975 (AD 009849).
8. Ames, S. R., J. and J. K. Millen, "Interface Verification for a Security Kernel," document in preparation.
9. Bell, D. E. and L. J. LaPadula, "Secure Computer Systems," ESD-TR-23-278, Volume I-III, The MITRE Corporation, Bedford, Massachusetts, November 1973-June 1974.
10. Graham, R. M., "Protection in an Information Processing Utility," *Communications of the ACM,* Vol. 11, No. 5, May 1968, pp. 365-369.
11. Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM,* Vol. 19, No. 5, May 1976, pp. 243-250.
12. Robinson, L., P. G. Neumann, K. N. Levitt, and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," *1975 International Conference on Reliable Software,* Los Angeles, California, April 1975, pp. 267-284.
13. Ames, S. R., "A Security Compliance Study of the Air Force Data Services Center Multics System," MTR-3065, The MITRE Corporation, June 1975.
14. Ames, S. R., Jr., "User Interface MultiLevel Security Issues in a Transaction-Oriented Data Base Management System," MTP-178, The MITRE Corporation, Bedford, Massachusetts, December 1976.
15. Lipner, S. B., "Comment on the Confinement Problem," *ACM Operating Systems Review,* Vol. 9, No. 5, May 1975, pp. 192-196.
16 Denning, D. E. and P. J. Denning, "Certification of Programs for Secure Information Flow," *Communications of the ACM,* Vol. 20, No. 7, July 1977, pp. 504-513.
17. Ames, S. R. and W. W. Plummer, "TENEX Security Enhancements," MTR-3217, The MITRE Corporation, Bedford, Massachusetts and Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts, April 1976.
18. Biba, K. J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford, Massachusetts, December 1976.

Network operating systems—An implementation approach*

by STEPHEN R. KIMBLETON, HELEN M. WOOD and M. L. FITZGERALD

National Bureau of Standards
Washington, D.C.

INTRODUCTION

Networking affords a communications support mechanism among heterogeneous computers. It does not provide any means for alleviating the effects of differences across systems. As a result, network users are forced to resolve such differences; the resulting learning and programming burden inhibits effective utilization of networking capabilities to achieve resource sharing.

Achieving the potential of networking would be increased through masking system differences from the user in many, if not most, applications. Network Operating Systems (NOSs)[1,2] are commonly viewed as the mechanism for accomplishing this objective. The functional objective of an NOS is to support and simplify access to existing services and to expedite the construction and subsequent accessing of new services by simplifying interaction among systems and between systems and users.

The remainder of the Introduction discusses the required enduser support which must be provided by a Network Operating System, identifies the two major resulting implementation issues and describes the constraints which affect the complexity of NOS implementation. Thereafter, the next section describes major NOS design decisions guiding the implementation of an Experimental NOS (XNOS) being implemented at the National Bureau of Standards (NBS) as part of a joint RADC/NBS effort. The third section then discusses the implementation of the user-system interface while the fourth section discusses the system-system interface. The fifth section provides some concluding remarks.

Network operating systems

A Network Operating System providing ease of access and utilization of systems, subnetworks, and services must provide two capabilities for endusers: job ececution and data handling.

A network job consists of a collection of network job steps, each of which may be executed on a different machine. The Network Job Execution function of a Network Operating System should provide the means for initiating job steps, migrating input information as appropriate, suitably disposing of output results and interacting with the user as required.

Network Data Support provides the means for a user to manipulate files using a command language and for a program to access remote records at run time. To provide a homogeneous viewpoint across systems, the NOS must provide a Network Wide Directory listing network accessible data resources, a common command language for manipulating files listed in this directory, and a means for limiting user access to these files to provide appropriate security mechanisms. At the record access level, mechanisms are required for preserving the meaning of data being transmitted between heterogeneous systems.

Implementation of these two enduser functions requires two major support functions: user-system interfaces which provides a common virtual viewpoint for the user across heterogeneous systems and system-system interfaces to support the transmission of data between systems and preserve the meaning of the transmitted data. Such interfaces are discussed in sections three and four, respectively. Their implementation requires resolution of some underlying design issues.

Design issues and alternatives

The difficulty of NOS implementation is critically dependent on the hardware and software environment which must be accommodated. Operationally, such situations can be divided into three categories: (i) those requiring a retrofit of the NOS to a heterogeneous collection of computer systems; (ii) those invovling retrofit to a homogeneous collection of computer systems; and (iii) situations in which there is no constraint to interface to existing software.

A study of these alternatives shows the following: (i) retrofitting to heterogeneous hosts is the most difficult case but maximizes resource sharing and best accommodates

* This work is a contribution of the National Bureau of Standards and is not subject to copyright. Partial funding for the preparation of this paper was provided by the U. S. Air Force Rome Air Development Center (RADC) under Contract No. F 30602-77-F-0068. Certain commercial products are identified in this paper in order to adequately specify the procedures being described. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best available for the purpose.

both planned and existing systems; (ii) retrofitting to a homogeneous collection of host computers is easier than retrofitting to a heterogeneous collection of hosts, but limits the opportunity for resource sharing—a major rationale for network construction; while (iii) unconstrained (e.g., built from "scratch") implementations will be the most satisfactory from a user viewpoint, but lack downwards compatibility and, as a result, will be extremely costly to implement since significant rewriting of existing software is required.[3]

NOS related projects

Before beginning our discussion of XNOS, we first briefly describe three NOS related projects: RSEXEC, NSW and ACCAT.

RSEXEC[4] was the first NOS oriented project. Since it was implemented for a homogeneous collection of computer systems (the TENEX systems within the ARPANET), the user-system interface problem was non-existent. Its primary contribution was exploration of the issues in providing a network wide file space and in resolving references to remote files. Such references were handed through an encapsulation mechanism termed a JSYS trap[5] which, upon trapping such a reference, determined where the file was actually located and arranged its migration to the site of the executing program.

The National Software Works[6] is designed to provide a sophisticated program production service. This is accomplished through networking together some of the more sophisticated program production tools available on different systems and, via NSW support mechanisms, making this collection appear as an entitiy to the individual user. Thus, upon issuance of a request to utilize a particular tool, the user and any appropriate files are automatically migrated to one of the sites at which the tool is available. As a result, the user only sees the provided tool and is unaware of the system on which it is being run. With the spectrum of accessible objects thus constrained, it proves feasible to provide a uniform treatment of error responses.

The Advanced Command and Control Architectural Testbed (ACCAT) is an interesting study in the problems of interfacing a user to a Database Management System (DBMS). A major problem in such an interface is the requirement that the user learn the syntax and semantics of the query language. To circumvent this, ACCAT contains a restricted natural language interface which transforms the user's request into an internal form. This form is then translated into a series of queries against the database.[7,8] Collectively, this provides a fairly "soft" interface of the user to the DBMS.

The preceding projects are primarily concerned with the utilization of networking capabilities to achieve specific functions. An explicit investigation of the interface issues implicit in supporting a general purpose NOS across heterogeneous hosts is being undertaken in the Experimental Network Operating Systems (XNOS) project at NBS. The functional components being investigated by these four projects are illustrated in Figure 1.

	USER-SYSTEM INTERFACING		NWD	SYSTEM-SYSTEM INTERFACING	
	CCL NJE	CCL FM		NETWORK IPC	RECORD ACCESS
RSEXEC	Y	Y	Y	N‡	N
NSW	Y*	Y*	Y*	Y*	N
ACCAT	N/A	N/A	N/A	N‡	Y
XNOS	Y	Y	Y	Y	Y

* JOBS LIMITED TO NSW TOOLS

‡ NOT USER ACCESSIBLE

Figure 1—Major NOS functional components

NOS IMPLEMENTATION

Implementation of a Network Operating System requires a series of design decisions. This section discusses two of the major configuration related decisions. The following two sections then articulate the issues in supporting the user-system and system-system interfaces and discuss implementation specific factors.

Support components

Construction of a Network Operating System requires both the design of certain procedures and processes, as discussed in the following two sections, and determination of the processors on which they are to be executed. Perhaps the major design decision relates to where computing requirements are to be provided. Two alternatives exist for providing these requirements: the existing hosts within the network (the unaugmented case) or, alternatively, adding specialized processors to satisfy most of these computing requirements (the augmented case) which we will call Network Interface Machines (NIMs).

Utilization of NIMs for satisfying most of the NOS computing requirements has four major advantages. First, it minimizes the impact of the NOS upon the host. This was deemed highly desirable since it is felt that few installation managers would tolerate any significant system overhead for NOS support. Even fewer would be willing to accept modifications of the host operating system.

The second advantage to offloading NOS support is the consequent minimization of the need to recode functionally identical modules to accommodate different systems and architectures. This approach also facilitates centralized design, implementation and support of the NOS. Hence, the third advantage—centralization—reduces costs by minimizing the number of support personnel.

The fourth advantage is facilitation of incremental expansion, modification and enhancement of NOS capabilities. This seems particularly important since evolving experience driven requirements are likely to result in gradual change of the initial NOS.

These four advantages must be considered in the context of two major implementation issues: attaching NIMs to hosts and the cost of providing the NIMs. After discussing the first issue and describing the initial XNOS implementation, we shall return to the second.

Attaching NIMS to hosts

Attaching a NIM to a host can be accomplished in two ways: implementation of the NIM as dedicated host(s) on the network or as a frontend processor to the host. To the authors, the second approach seems appropriate when Network Operating Systems have achieved a greater degree of maturity. A significant amount of work has been concerned with the relationship between the host and the frontend.[9] The primary concern of those studies is with optimization of the provided functions rather than with identification of the functions to be supported. Since present research is more appropriately oriented toward identification, realization of the NIM as a dedicated host(s) on the network allows concentration on NOS support issues and therefore seems preferable.

A major advantage of implementing the NIM as a dedicated computer attached to the subnetwork is the consequent utilization of the host-host protocol to support communication between NIM and hosts. The disadvantage is forgoing the simplification of connecting hosts which are ill suited to interactive use.

The initial implementation—XNOS

An Experimental Network Operating System (XNOS) has been implemented at the National Bureau of Standards to facilitate investigation of the interface issues (discussed in the following two sections) which must be supported to achieve more effective utilization of network accessible resources. Figure 2 illustrates the initial XNOS configuration and shows that NOS support is to be provided through a NIM attached to the communications subnetwork (the ARPANET).

Given that the NIM is to be a dedicated host on the communications subnetwork, only one is required to support a full feasibility demonstration. This follows since ARPANET technology permits a host (the NIM in this case) to establish a network connection with itself. Thus, a single NIM can provide the functions required to interface to a host issuing a request, the functions required to interface to the host responding to the request, and the communications functions required to interface between two NIMs. Thus, the basic implementation approach illustrated in Figure 2 provides a reasonable, general purpose test environment.

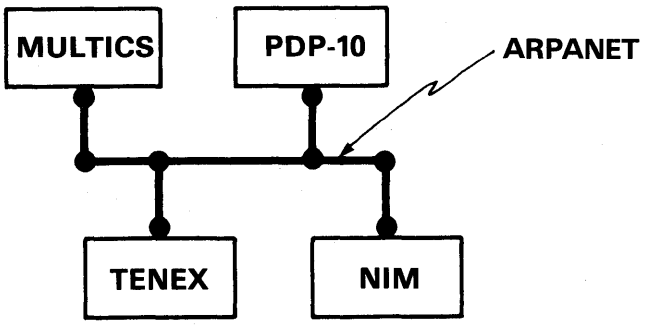


Figure 2—Initial XNOS configuration

The preceding comments have provided a general perspective on the adopted XNOS implementation approach. In this environment, the NIM is a PDP-11/45 running the UNIX operating system and attached to the ARPANET via a slightly modified version of the University of Illinois Network Control Program (NCP).

The XNOS supported hosts are: a Honeywell 6180 running the MULTICS operating system, a PDP-10 running TOPS-10, a PDP-10 running RENEX, and a PDP-11/45 running UNIX. Based upon present experience we foresee little difficulty in supporting any interactive system attached to a communications subnetwork, preferrably using full duplex communications. Support of batch systems has not yet been investigated.

Logging in

Two basic approaches to logging into XNOS can be considered. In the first approach the user is directly logged into a host computer and invokes XNOS support software thereby being indirectly logged into XNOS. In the second the user directly logs into the NIM (and hence directly into XNOS). The first approach potentially allows the network user to see the network as an extension of the host and to use the same command language used by that host computer.

Adoption of the second approach seems more natural from an NOS viewpoint since it permits equal treatment of all hosts. Thus the user physically logs into the NIM, issues commands to the NIM receives responses from the NIM and, in general has the NIM acting as a mediator between the user issuing the requests and the systems satisfying them. This approach also simplifies support of utility functions.

Utility support

Effective support of network users is facilitated through the existence of certain support capabilities, e.g., editors, that function in an identical manner regardless of the host to which one is attached. XNOS supports this requirement by providing such utility functions on the NIM. Invoking these functions requires that any required input data or files also be located on the NIM. This is accomplished through using the mechanisms in sections three and four.

NIM computational requirements and costs

The cost of providing NIMs is dependent upon their computational requirements and the number provided. Such

costs must be carefully considered since the canonical problem in providing any type of frontend support to a host computer is the tendency for such support to require another 'host' of size comparable to that being supported. We feel that this is not a problem of XNOS since the only functions provided by the NIM are those supporting user-system and system-system interfacing together with utilities.

Computational requirements

The computational requirements for supporting user-system interfacing can be divided into: (i) support of the common command language for file manipulation and the network job execution command, and (ii) support of NIM independent interactions. Typically, for an individual user, the average time between issuing commands in the first category would be on the order of minutes. The second category of interaction occurs when no intervention by the NIM is required, e.g., the user is interacting with a program, and the only computational support is that of forwarding commands or responses. We assume that the overhead in handling such commands is minimal.

The discussion in section four shows that the computational requirements for supporting system-system interfacing are those required to provide a network wide Interprocess Communication (IPC) mechanism and those required to support translation and transformation of records generated by remote record accesses.

IPC is used to support both user-system and system-system interfacing. As discussed above, the former is invoked relatively infrequently. We conjecture that the mean time between occurrences of remote record accesses by an individual program will also be on the order of minutes due to the subnetwork delays which occur in accessing remote hosts. Moreover, computational support of the translation and transformation requirement generated by a remote record access is relatively minor as follows from the discussion in section four. (These estimates are only intended to indicate that NIM computational requirements are likely to prove acceptable. Their verification requires measurements which can only be performed after a significant level of user utilization of XNOS capabilities.)

Utility support, particularly editing, can require a significant fraction of system resources. However, such support offloads the host of a function which it could provide (for interactive systems). Since this is done, in part, as a convenience for the host, the requirement for supporting utility functions should not be directly counted against NIM computational requirements.

Cost considerations

The cost of providing NIMs is dependent upon the number supported and the size of each. Assuming that NIMs are implemented as hosts on the communications subnetwork, the minimum number of NIMs is one. A likely upper bound is one for each host. In an operational version of XNOS, the actual number used will probably be heavily influenced by a combination of performance considerations and the opportunity which NIMs afford for offloading certain utility functions (such as editing) from the hosts. A reasonable initial estimate for an operational environment would be one NIM per operating system class.

Utilization of a NIM promises to become increasingly cost effective if declining hardware costs are balanced against increasing software implementation costs. Even now, a relatively powerful NIM can be procured for the equivalent of the burdened cost of supporting one professional programmer. Moreover, since this approach minimizes reimplementation of functionally similar modules and facilitates modification, one must also balance the one time cost against the continuing programmer costs.

Utilization of a NIM also permits offloading certain terminal support functions from the mainframe. As a result, system efficiency is potentially improved and systems which are ill suited to interactive utilization are made more usable in a networking environment. Of course, the relation between the host and the NIM is an important factor in any final decision.

THE USER-SYSTEM INTERFACE

A Network Operating System should present a standardized view of network resources to its users and should provide ease of access to those resources. This section describes an implementation approach for a user-system interface that provides these functions for the XNOS.

Implementation requirements

The network user should be able to view network resources in the same manner that a user would view an individual system. A user knows of files in his workspace, but generally is not concerned with the complexities of the physical device upon which they are located. Likewise, the network user should be able to know there are files in his (network) workspace which exist on the available systems of the network, but should be insulated from the intricacies of the physical systems upon which the files reside. In addition, since the network user would like to view the network as an entity, a common command language is desirable to support communication both with individual hosts and across hosts.

These capabilities are provided within XNOS by a Networkwide Directory System (NWDS) that describes the user, the user's files and the available systems of the network. The NWDS also provides a common command language facility for manipulating known files and programs.

The Networkwide Directory System (NWDS)

The NWDS consists of (i) a data base, the Network Wide Directory (NWD), that describes the files belonging to each

user, the hosts on the network, and the types of interactions allowed among those entities; (ii) the software support required to access the NWD data base; and (iii) the access control mechanisms that insure system integrity and data security.

New users to the network are queried by the NWDS before entries are made into the NWD data base. These entries identify the user and the systems to be accessed. The user's own network directory is then initialized and all connections to hosts are made on behalf of the user by the NWDS. Thus, the user sees the network as a large relatively integrated system composed of the resources available to the user on each of the accessed hosts.

From the user's viewpoint, files have no rigid naming conventions. To accommodate individual system constraints, the NWDS maps these user file names into unique host file names conforming to the requirements of the host on which the file resides. Manipulation of these files is preformed by the NWDS for the user in accordance with user commands.

In addition to providing a common command language for the user, the NWDS contains logical record descriptions and host representation tables to support preservation of meaning in transmitting records between heterogeneous hosts. This supports run time access by a program to records contained in remote files as discussed in the following section.

User commands

The NWDS provides a common command language for file manipulation and a command for executing network jobs. This section describes these commands and presents an example of Network Job Execution.

Common command language for file manipulation

XNOS file manipulation capabilities provide a common command structure for manipulating files residing on an individual system and for transferring files among systems. The chosen set of commands was developed by surveying file manipulation commands available on the systems currently supported by XNOS.

Determination of the precise set of commands to be supported requires some care since support of a 'network' viewpoint of host resources is inconsistent with requiring detailed local host knowledge of how files are actually mapped to devices. As a result, only commands dealing with files as logical entities were selected for implementation as listed in Table I. Note that not all of the systems provided all of these commands. Thus, the NWDS effectively extends system capabilities.

TABLE I—The Common Command Language

File Handling Commands	
APPEND	FILE1 FILE2
COMPARE	FILE1 FILE2
COPY	FILE1 FILE2
CREATE	FILE
DELETE	FILE1
ERASE	
LIST	
RENAME	FILE1 FILE2
TYPE	FILE
UNDELETE	FILE
File migration Commands	
RETRIEVE	FILE HOST
TRANSFER	FILE HOST

Network job execution

A network job consists of a collection of network job steps. Each job step specifies a program, the host on which it is to be executed, input data files and output data files. Prior to initiation of the job step all input files are checked to ensure that they are resident at the site where the program is to be executed. If not, appropriate transfer operations are begun and step execution is deferred pending completion of these transfers. The syntax for a job step is:

```
RUN PROG@HOSTA
IN1=INPUT1,IN2=INPUT2,. . .
OUT1=OUTPUT1@HOST1,OUT2=OUTPUT2@HOST2. .
```

where

PROG is the program to be executed
HOSTA is the execution site
INn are the logical names for input data
INPUTn are the names for the input data files
OUTn are the logical names for output data
OUTPUTn are the names for the output data files
HOSTn are the final locations for the output data

A network job and its job steps are defined by an interactive program used in conjunction with the NWDS and the commands described above. The program queries the network user to gather all the information needed to specify a network job. For each job step in the network job, the user enters the program name, execution site, and for each input or output file, the logical name (name expected by the program) and the actual name (name in the user's directory). The program builds a macro that invokes the system interactions necessary to perform the job step. The resulting command, called RUN, is issued like any other command and the network job is executed. Figure 3 indicates a sample session defining a network job.

Access controls

Acceptance of a Network Operating System requires access controls to limit the access of the user to programs and files and to limit the access of a program. User level access controls are provided through a profile capability which contains a complete description of the network user, the

The user wishes to execute the program TEST on the NBS PDP-10 (TEST@NBS10). There are three input files, FILE1, FILE2 and FILE3 which are to be given the logical names IN1, IN2 and IN3 respectively. The program produces three output files: OUT1, OUT2 and OUT3. They are to be renamed and moved to other sites as follows: DATA1 goes to NBSUNIX, DATA2 goes to BBN and DATA3 will remain at the NBS10. There will be only one Network Job step in this job. A carriage return indicates no response.

The following dialogue will take place between the user and the XNOS:

```
Enter Program Specifications : TEST@NBS10
Remote file access? (y or n) n
Logical Name for Input File 1: IN1
Actual Name for Input File 1: FILE1
Logical Name for Input File 2: IN2
Actual Name for Input File 2: FILE2
Logical Name for Input File 3: IN3
Actual Name for Input File 3: FILE3
Logical Name for Input File 4:
Logical Name for Output File 1: OUT1
Actual Name for Output File 1: DATA1
HOST 1=NBSUNIX
Logical Name for Output File 2: OUT2
Actual Name for Output File 2: DATA2
HOST 2-BBN
Logical Name for Output File 3: OUT3
Actual Name for Output File 3: DATA3
HOST 3=
Logical Name for Output File 4:
```

Figure 3—Sample Definition of a Network Job

systems he is permitted to access and the types of interactions allowed.

Program access controls are supported in two ways: (i) run time checking of individual data items to ensure that the program has appropriate authorization, and (ii) execution of the program in its own run time directory to limit references by the program in a manner commensurate with the information provided by the user in issuing the job step execution command.

In addition to these formal access control mechanisms some degree of additional protection is provided by having file names in the directories on individual hosts (Local Host Directories) which differ from the names by which the files are known to the user. Although the individual user will be permitted to determine host file names corresponding to those files in his directory under certain conditions, such information is not available for files belonging to other users.

Support of such access controls requires that the NOS be the "owner" of all files resdient on an individual system. However, to facilitate interactions by a user wishing to directly manipulate files, temporary directories can be set up which contain those files designated by the user and which support direct access by the user rather than access mediated by XNOS.

Implementation support

The common command language described above was implemented using the capabilities provided by the Network Access Machine (NAM)[10,11] developed at NBS. The NAM is implemented on a PDP-11/45 minicomputer running the UNIX operating system and provides directives which permit a user to interact with a remote system on a network. Sequences of these directives stored in files called macro files have been written that will generate the necessary machine-dependent dialog to perform file manipulation functions required for each of the host systems. Thus, execution of each command in the language is actually a call to a macro file that the NAM expands and executes.

The responses that the remote system generates are, in turn, analyzed by the NAM. By permitting the mapping of the varied responses from the remote systems into standardized messages for the network user to see, the NAM facilitated the commonality of the common command language. The user can enter a standard set of commands and see a standard set of responses, and need not become familiar with the responses of each system accessed. (Note that this approach supports standardized error messages across all anticipated error responses; unanticipated responses are transmitted directly to the user.)

SYSTEM-SYSTEM INTERFACING

Successful system-system interfacing within a Network Operating System environment requires two primary functions: (1) a means for transmitting data between systems, and (2) a mechanism for accessing and preserving the meaning of structured data as it is transmitted across heterogeneous systems. The first feature is provided by Interprocess Communication (IPC), and the second by a Remote Record Access (RRA) capability.

In this section we shall discuss the problems of system-system interfacing in more detail and identify required functional capabilities and support mechanisms, while considering some of the alternatives encountered during the implementation of XNOS. A more detailed treatment of the problem may be found in Reference 12.

Interprocess communication

Interprocess Communication (IPC) provides the basic mechanism for initiating and controlling the flow of data between cooperating processes. Since processes are the only active entities within a computer system, IPC is a basic building block for supporting communication between computers.

The National Software Works[6] has implemented a relatively sophisticated IPC mechanism termed MSG[13] which permits the SENDing process to continue execution independently of the activities of the RECEIVEing process, whether or not the message was actually RECEIVEd. Providing such a capability requires fairly complex software support.

The XNOS IPC mechanism might be regarded as a minimum level IPC approach requiring very little host software to support its functioning. This is accomplished using a CALL/RETURN based mechanism that functions analogously to subroutine CALLs in that the WAIT state is entered upon issuance of the CALL. The simplicity of imple-

menting this approach stems from using constructs available in the major existing programming languages and requires minimal host support.

The simplicity of XNOS IPC support derives from the fact that, if we assume interactive jobs, IPC messages can be transmitted over the controlling teletype channel, intercepted at the NIM and diverted from the user's terminal to the appropriate destination where the reverse process occurs. Although the sophistication of this approach is limited and the bandwidth between communicating processes is constrained, the simplicity of its support coupled with the negligible startup costs for any host supporting interactive processing have led to its adoption.

Remote record access

Remote record access permits a process executing in one computer to access data records located in another computer at run time. This requires transmitting data between possibly dissimilar hosts which is complicated by both physical incompatibilities between systems and organizational incompabitilities between the data as required/maintained by the systems.

Physical incompatibilities include differences in word length, character packing, sign bit location, and data encoding (e.g., character sets or use of two's vs. one's complement representation of numeric data). A translator is needed to convert data from the internal format of one host to that of another. Organizational incompatibilities between data bases, on the other hand, result in the need for a transformer to modify the format of files and data items (e.g., reorder and concatenate the data fields of a record before transmitting that data to the requesting host).

The provision of a remote record access capability, then, requires: (1) a mechanism for selecting a record from the file/data base containing it, (2) a record translator to preserve meaning in transmitting the record between dissimilar hosts, and (3) a record transformer to permit the alternation of record structures. Supporting these functions mandates a means for describing the physical and organizational characteristics of the hosts. (It should be noted that the same functions could be performed iteratively for all records, when an entire file is migrated between dissimilar hosts—thus resulting in a file transfer protocol (FTP) for structured files.)

These functional components will now be discussed in greater detail and, to better illustrate the associated information requirements, an example will be described in which a process on one host requests a data record from a dissimilar host on the network. As illustrated in Figuare 4, the host bearing the file containing the remote record is called the Data Host (Dhost) and that bearing the accessing program is the Process Host (Phost).

Record selection

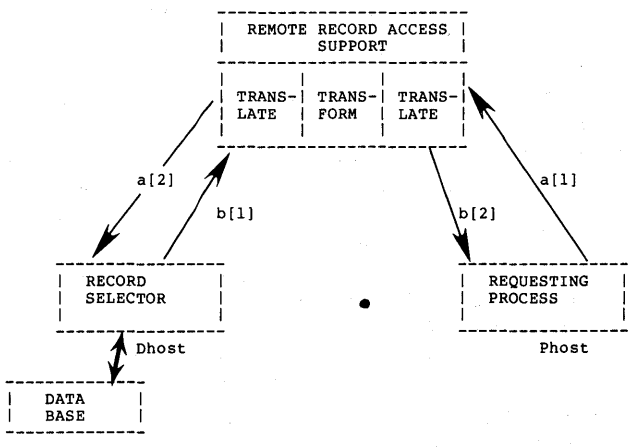
The precise mechanism which supports record selection is dependent upon capabilities existing at the host computer including those provided by a data base management system, if any. The following discussion assumes the existence of a suitable mechanism for retrieving a record based on utilization of a unique key, if random access techniques are being employed or, alternatively, the keyword 'next' if sequential access is being used. Given this assumption, we now describe the basic communication support.

Assume that the Phost issues a request for a record located on the Dhost (path a[1] in Figure 4). This request, when received by the NIM, is forwarded to the Dhost (path a[2], Figure 4), accompanied by any additional descriptive information needed by the selector. Such information would include the host specific file name, as determined by the Network Wide Directory System.

Upon receipt of the request by the Dhost, a record selector is responsible for retrieving the record and transmitting it to the record translator (path b[1], Figure 4). Figure 5 shows such a record, in the format maintained by the Dhost. (The field names are included here for clarity, but would usually not be contained within the actual record.) Although the record in Figure 5 shows all data fields in character notation, it should be understood that these data items are actually in the machine representation of the appropriate data type (e.g., integer, real, logical, character, binary).

Record translation

Record translation preserves the logical record structure and data element type (e.g., real, binary, logical, integer, character) and, for arithmetic data elements, precision. Thus the record translator must know the exact format of the record to be translated down to the data item—a level of detail analogous to that contained in a FORTRAN format statement (e.g., 3I5,2X,5A5,F4.2). In addition, the internal



a[1]: request for data from network-wide data base
a[2]: request for data with access parameters
b[1]: retrieved record in Dhost format
b[2]: retrieved record in Phost format

Figure 4—Remote record access components

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------

|SSN      NAME                 BIRTH  ALRGYTEST HT
|-------------------------------------------------
|                                      0A00001000
|333775555CHARLES J SMITHSON  10269200200F100030182

|  WT SDISEASE1       DATE   MEDICATION     DISEASE
|-------------------------------------------------
|88075MMALARIA        0601943QUININE        GOUT

|2       DATE   MEDICATION     DISEASE3       DATE
|-------------------------------------------------
|        0801965CORTISONE      STREP          11039

|  MEDICATION     DISEASE4       DATE   MEDICATION
|-------------------------------------------------
|69PENICILLIN     ANGINA PECTORIS0703972DIGITALIS

|    MOTHER              ADDRESS
|-------------------------------------------------
|    REBECCA B SMITHSON  BOX 444, TAMPA  FLA 00000

|              FATHER              ADDRESS
|-------------------------------------------------
|              FREDERIC SMITHSON   DECEASED

|                        YCYMDFILLER
|-----------------------------------
|                        1020T
```

Figure 5—Data host record—DDR[D]*

*The characters in the allergy-related fields are in hexadecimal representation. The upper digit represents the first four bits of an eight bit binary field, while the lower digit represents the remaining four bits of that field.

format of all data types must be known for each computer that is supported on the network.

Placement of the record translator at the host requires its coding for each separate host architecture and increases the impact of the network on the host. For both of these reasons, the record translator has been implemented on the NIM. Although knowledge of the internal data representations of N hosts are still required, only 2N, rather than N(N-1) translators must be implemented (one into and one out of each host format). Since host dependencies are reflected only in the tables being used to drive the translation process, 2N becomes 2. As the process of translating into and out of host dependent representations are very similar, it was initially conjectured that 2 would become 1. However, there prove to be enough subtle differences to justify two separate programs rather than consolidating them into one.

Given that translation is to be supported on the NIM, the remaining issue is whether to map directly from source to destination form or to proceed via a common intermediate form. The substantial simplification in implementation afforded by the intermediate Network Normal Form (NNF) resulted in the selection of this approach. Figure 6 shows the Dhost record as it would appear after translation to NNF format. (Notice that all data types are now in a character representation.) Once the record has been transformed to meet the needs of the requesting process (Figure 7), it is again translated—this time into the bit representation required by the Phost, a process that is essentially the inverse of the original, table-driven translation from Dhost format

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------

333775555/CHARLES J SMITHSON/1026920/+000000001010
0010000000000000000000001111/+0000000100010000000
0000000000000000011/+182.88/+75/M/MALARIA/0601943
/QUININE/GOUT/0801965/CORTISONE/STREP/1103969/PENI
CILLIN/ANGINA PECTORIS/0703972/DIGITALIS/REBECCA B
 SMITHSON/BOX 444, TAMPA FLA 00000/FREDERIC SMITHS
ON/DECEASED/+10/+20/*T*/
```

Figure 6—Data host record DDR[NNF]

to NNF. Figure 8 shows the record in the Phost format. The resulting record is then transmitted to the Phost (path b[2], Figure 4) and RETURNed to the requesting process.

The communications subnet often supports character translation. Therefore it would at first seem reasonable to have the translation of character data handled by the subnet, thus reducing the translation requirements to a minimum. However, as it is unreasonable to assume the exclusive use of character-oriented data, such subnetwork capabilities cannot be relied upon for the general case. It may be desirable, however, to add to XNOS a capability to default to such subnetwork handling of data when dealing with character-only data.

Record transformation

Record transformation supports modification of both the logical structure of the record and individual data elements. The record transformation function matches the information transmitted to the needs of the receiver or to ensure proper protection of sensitive information (e.g., matching the data to the access rights of the receiver by omitting sensitive information from the record before transmitting it on to the requesting process). Such transformation affects the logical structure of the record through one of three basic transformation types: logical, arithmetic, or string.

Logical transformations such as AND or OR generate Boolean binary strings resulting from the bit by bit ANDing and ORing of two successive strings. Arithmetic transformations such as: $+,-,/,\times$ act as would be expected. String transformations can be quite complex as evidenced by the

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------

333775555/CHARLES J SMITHSON/M/+50/1026920/+000000
0000000000000000000000000000000011/+00000001101100
1000000000000000000000001111/+165/+6.1/STREP/1103969
/PENICILLIN/ANGINA PECTORIS/0703972/DIGITALIS/*T*/
```

Figure 7—Phost data record—PDR[PNNF]

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------

|SSN      NAME                 SYGBIRTH  TALRGSALRGW
|-------------------------------------------------
|                                        000000B000
|333775555CHARLES J SMITHSON  M3010269200000312000F1

|T HT DISEASE1       TREATMENT      DATE   DISEASE2
|-------------------------------------------------
|65604STREP          PENICILLIN     1103969ANGINA P

|       TREATMENT      DATE   DFILLER
|-------------------------------------
|ECTORISDIGITALIS      0703972T
```

Figure 8—Phost data record—PDR[P]

capabilities of string manipulation languages. Initially, a concatenation capability seems desirable. Among the additional transformations that may be needed are algorithms for the compression and/or decompression of textual information.[14]

As with the record translator, support of record transformation is implemented more efficiently on the NIM. This follows through having the transformer operate on the record in NNF format.

In XNOS a Transformation Description Table is used to provide the information that the translator needs to establish relationships between data items in the Phost and Dhost records. In this table each item in the Phost data record (PDR[i]) is expressed as a function of the appropriate item(s) in the Dhost data record (DDR[j]). For example, the concatenation of two character fields from the Dhost record might be expressed as follows:

$$PDR[j]=DDR[i] \mid DDR[k]$$

A record that could be the result of such a series of operations is shown in Figure 7.

Supporting transformation of numeric data types (i.e., integer, real) requires care since precision must be considered. Our approach uses a character based intermediate representation coupled with an arbitrary precision arithmetic package. The character representation directly accommodates varying precision while the arbitrary precision package supports transformations of numeric types. Of course, the precision of the result will never be better than the lesser of that of the source and destination. However, this approach does ensure that precision is not lost during the translation/transformation process.

Secure environments

The enhanced access to system resources provided by a Network Operating System requires a suitable collection of access control mechanisms. Not only must a user's identity be verified at login time, but access on his behalf to NOS maintained accounts on remote systems and to files, records, and fields within those systems must also be controlled.

As part of a continuation of the joint NBS/RADC effort which resulted in the work described in this paper, efforts are currently under way to structure NOS access control requirements and alternatives. This study will result in the incorporation of prototype access control components into XNOS. Since the password has been found to be a relatively inexpensive, but highly effective mechanism for identify verification and access control when used properly[15,16] it will no doubt be an integral part of the effort to enhance XNOS security.

Once the decision has been made to control access to network resources, the ability to protect data during transmission becomes a further requirement. Although it is generally accepted that an algorithm such as the Data Encryption Standard[17] would be suitable for such purposes, an appropriate encryption key management mechanism would have to be determined for the NOS environment. These and other network security related issues will be addressed in subsequent studies and reports.

CONCLUDING REMARKS

In this paper we have provided a perspective on the issues and implementation constraints implicit in providing a general purpose Network Operating System for a heterogeneous collection of host computers. One implementation of an NOS has been described based on utilization of Network Interface Machines to offload NOS support from the host and to facilitate centralized design, implementation and support.

Based upon our experience in the implementation of XNOS we are led to conclude that NOSs are feasible. However, in view of the complexity of the possible error messages, the activities of the general purpose user of such a capability will probably be constrained to minimize unexpected error responses. Thus, we would anticipate that general purpose computing might be confined to one or a few machines while other systems would be accessed to use unique resources known to be fairly well debugged.

Finally, it should be noted that Network Operating Systems only solve part of the problem of simplifying user access to systems—that of providing a uniform viewpoint across systems. The remaining part is concerned with simplifying system access given such a capability and constitutes the area of network access support. In Reference 18 it is shown that such support can be structured into four major components: user and service profiles to simplify and automate user interaction with services; soft user interfaces to eliminate the need for user concern with the precise syntactic structure of commands; expert assistance to reduce the need for entering repetitive information; and dynamic tutorial assistance to provide specific, pinpointed problem solving information online.

REFERENCES

1. Kimbleton, S. R. and R. L. Mandell, "A Perspective on Network Operating Systems," *Proc. 1976 National Computer Conference*, AFIPS Press, Montvale, New Jersey, Vol. 36, 1976, pp. 551-559.

2. Forsdick, H. C., R. E. Schantz, and R. H. Thomas, "Operating Systems for Computer Networks," BBN Report No. 3614, Bolt Beranek and Newman, Cambridge, Mass., 1977.
3. Kimbleton, S. R., "Network Operating Systems: Issues, An Approach, and Some Conclusions," in preparation.
4. Thomas, R. H., "On the "Design of a Resource Sharing Executive for the ARPANET," *Proc. AFIPS 1972 National Computer Conference,* Vol. 42, AFIPS Press, Montvale, New Jersey, 1973, pp. 155-164.
5. Thomas, R. H., "JSYS Traps—A Tenex Mechanism for Encapsulation of User Processes," *Proc. 1975 National Computer Conference,* AFIPS Press, Montvale, New Jersey, 1975, pp. 351-360.
6. "National Software Works," Semi-Annual Technical Report, Massachusetts Computer Associates, Inc., Wakefield, Massachusetts, 1976.
7. Morris, P. and D. Sagalowicz, "Managing Network Access to a Distributed Database," *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks,* May 1977, Berkeley, California, pp. 58-67.
8. Hendrix, G. G., "Lifer: A Natural Language Interface Facility," *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks,* 1977, Berkeley, California, pp. 196-201.
9. Padlipsky, Michael, "A Proposed Protocol for Connecting Host Computers to ARPA-like Networks via Front-end Processors," Network Working Group, RFCNo. 672, NIC 31117, TIP, Oct. 1974.
10. Rosenthal, Robert, *A Review of Network Access Techniques with a Case Study: The Network Access Machine,* NBS Technical Note 917, July 1976, 36 p.
11. Rosenthal, Robert, and Lucas, Bruce D., *The Design and Implementation of the National Bureau of Standards' Network Access Machine (NAM),* NBS Special Publication Series 500, in preparation.
12. Wood, H. M. and S. R. Kimbleton, "Remote Record Access: Requirements, Implementation, and Analysis," in preparation.
13. Rivas, L. P., H. C. Ludlam, and R. T. Braden, *An Implementation of the MSG Interprocess Communication Protocol,* Campus Computing Network, ARPA Order No. 2543-3, May 1977, 128 p.
14. Schneider, G. M. and E. J. Desautels, "Creation of a File Translation Language for Networks," *Information Systems,* 1975, Vol. 1, pp. 23-31.
15. Wood, *The Use of Passwords for Controlled Access to Computer Resources,* National Bureau of Standards, Special Publication 500-9, May 1977, 79 refs.
16. Wood, Helen M., "The Use of Passwords for Controlling Access to Remote Computer Systems and Services," *Proceedings of the National Computer Conference,* AFIPS Press, 1977, pp. 27-33, 43 refs.
17. National Bureau of Standards, *Data Encryption Standard,* FIPS PUB 46, January 1977.
18. Watkins, Shirley Ward, and Stephen R. Kimbleton, "Network Access Technology: A Perspective," in preparation.

A distributed processing system for naval data communication networks*

by WESLEY W. CHU and DAVID LEE
University of California
Los Angeles, California

and

BRANDON IFFLA
Naval Oceanic System Center
San Diego, California

INTRODUCTION

With the advent of microcomputers and computer networks, distributed processing (i.e., the sharing of computing by several processors with each processor assigned to perform a certain task) becomes technically and economically feasible. Such architecture also provides system expandability, system reconfiguration, and fault tolerant capability for the system. The performance of such a system is not only affected by its application but also by the system architecture.

One of the important problems that affects system performance is interprocess communication. It is intimately related with the task partition and assignment of various process modules and the system bus structure that provides the interprocess communications. In this paper we will first describe a new system architecture which consists of modules with identical bus interface connected by a unique broadcasting bus structure. In order to reduce bus interface cost, serial broadcasting busses are used. Next, we describe the simulation model for studying the performance of the proposed architecture. Finally, we use the known program characteristics and workload profile of a Naval data communication network as an example of the proposed architecture in order to study its feasibility and performance. The quantitative relationships among process and bus utilization, message delay, and task partitioning and assignments of the distributed processing system are presented. These results provide us with insight and understanding of the behavior of the distributed processing system.

THE DISTRIBUTED PROCESSING SYSTEM ARCHITECTURE

The proposed distributed processing system as shown in Figure 1 consists of three module types: processor, memory and input/output (I/O) modules. Modules may communicate with each other via a unique broadcasting bus structure. Each module has an identical bus interface consisting of a serial send bus and several receive busses. The send bus transmits data serially to all other modules in the system. The receiver receives data serially on its receive busses. For such a bus organization, a system with N modules requires N busses to communicate among the modules.

Every module has an identical hardware bus interface (BI) and a front end processor (FEP). The FEP handles data transfers and protocol between the bus and the module, coordinates data transfers, discriminates between message types, and handles bus transactions for all modules. The FEP of a processor module acts as an input/output controller for the main processor. When it is implemented as a microprocessor or as a bit-slice processor, it provides a reliable and inexpensive approach to handling data transmission and bus protocol. It also provides an inherent distributed system intelligence which can be programmed to control the allocation of hardware resources. A microprocessor is used as the main processor of the processor module.

Data enters the system via the I/O module. The I/O module may store the data temporarily in its own FEP local memory, or it may transfer data directly to any other module via the bus. Similarly, the FEP of a processor module may store data received from the system bus temporarily in its own local memory or transfer it directly to the main processor local memory. The FEP of the memory module may accept data temporarily in its own local memory or transfer data to the common bulk memory. Any module can initiate system output by transferring data to an I/O module.

The processor module

The processor module organization shown in Figure 2 assumes a microprocessor or bit slice processor implementation. The module consists of a main processor, an FEP, local memories, a Direct Memory Access (DMA) facility, a

* This research is supported by the U.S. Office of Naval Research, Contract no. N00014-75-C-0650.

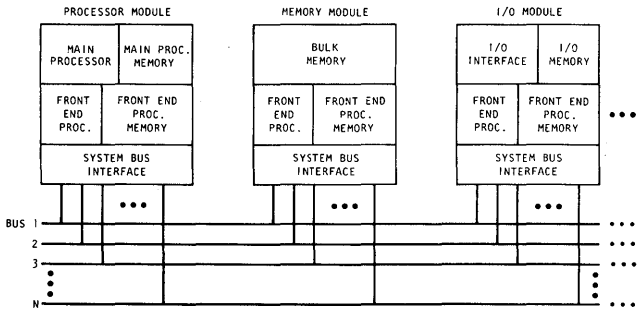


Figure 1—A distributed processing system architecture

bidirectional processor to processor interface and a system bus interface. The main processor executes the application program. The main processor local memory consists of read only memory (ROM) for program storage and random access memory (RAM) for data storage. The FEP has a small amount of local memory programmed to handle bus I/O and bus protocol. Local memory is dedicated to each main processor for program storage to alleviate contention by several processors accessing common memory, thereby reducing the inter-module communications. The FEP transfers data to and from the bus, as relayed to it by the system bus interface, and writes the data in the main processor local memory through the direct memory access. The reverse process occurs when data is read from the main processor local memory.

The DMA facility allows the FEP to store data in and retrieve from the RAM of the main processor. This interface ensures that data transfers are transparent to the interrupt and I/O control hardware of the main processor. The processor to processor communication interface allows word transfers to take place directly between the main processor and FEP without both accessing the main processor local memory. Either the main or the FEP can interrupt the other directly through their respective I/O facilities. The FEP receives or transmits data from or to system busses and provides asynchronous communications to the main processor, I/O control facilities, or to the main processor local memory.

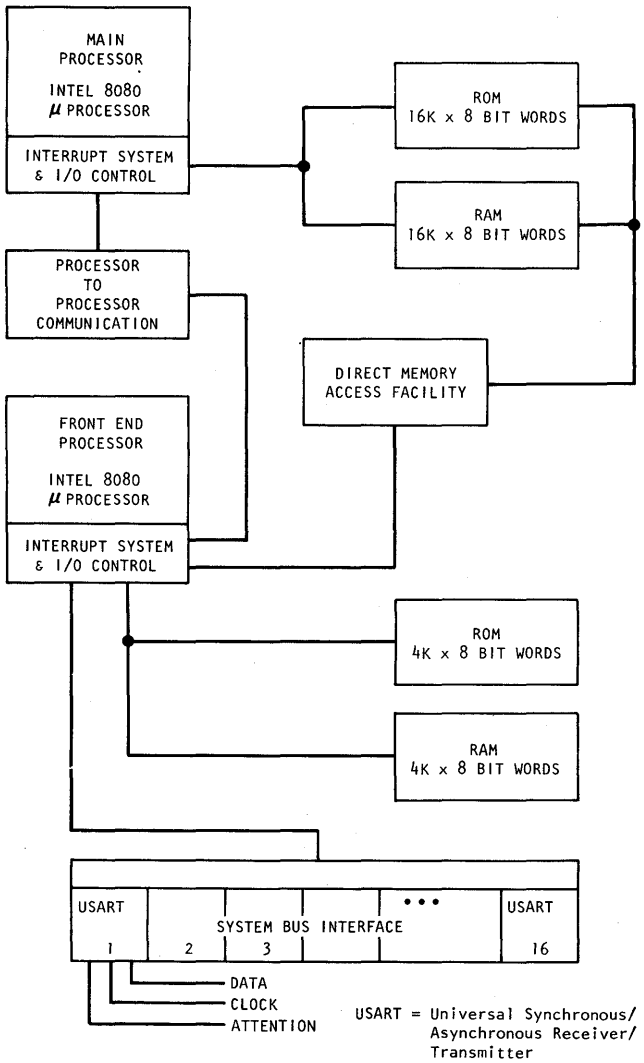


Figure 2—An organization of the processor module

The broadcasting bus structure

The proposed system may be viewed as a network consisting of different types of modules each having its own dedicated send bus which broadcasts information to all other modules. Further, each module can "listen" to the information from all the other modules in the system. Bus bandwidth affects the delay in inter-module communications; a serial broadcasting bus structure is used to both increase the bandwidth and reduce the bus and bus interface complexities.

Further, faults generated by hardware failures in such bus organizations are not likely to affect more than one bus, thus providing a graceful system degradation if software allows. The proposed bus structure can also be arranged to transmit parallel data thereby increasing the bandwidth for inter-module communications. However, system bus reliability decreases as the bus hardware complexity increases, and increasing the number of bus lines also increases the costs of the system bus interface. Hardware allocation schemes for bus hierarchies inherently become more complex as the number of busses increases. For real time applications, contention for a bus among modules is a severe problem when the contention time exceeds a specified time requirement for the system to respond to an external interrupt. Broadcasting bus organizations alleviate bus contention problems.

For the proposed bus organization, a system with N modules requires N busses to communicate among the modules. The system bus interface of each modules consists of N individual bus interfaces. Each bus interface (one per bus) decodes control signals from the FEP, performs serial-to-parallel data conversion, encodes and decodes character parity checks, and stores data for transmit or receive. Commercially available LSI components such as Universal Synchronous/Asynchronous Receive Transmitter (USART) or

Intel microcomputer (e.g. 8048) may be used for such interfaces.

Inter-module communications

Communication between modules is initiated by a module (called the source) transmitting a header, a character, or a block of data at a time. The bus interface of each receiving module of the system receives and temporarily stores the character of data and interrupts its FEP. The FEP of the receiving module honors the interrupt by resetting the interrupt facilities in the bus interface and then reading the identification of the bus request. The FEP of the receiver reads and decodes the destination address. The source begins transmitting data after receipt of an acknowledgment from each of the designated receivers. The receiving FEP disables the bus communication by not resetting the interrupt facility if the data is not addressed to it or if it does not desire to receive the data at that time. The receiving FEP records the time when the transfer request character was recognized. Since a receiver must send an acknowledgment honoring the transfer data request within a specified "time out" period, the source waits to receive an acknowledgment before it begins sending its block of data. All data communication on the bus is in block format. Asynchronous data communication requires DATA and ATTENTION lines for each bus. Synchronous data communication requires DATA, CLOCK, and ATTENTION lines for each bus (Figure 2). The receiving FEP takes one character of data at a time from the bus interface and stores the data temporarily in its own local memory or stores the data, via the DMA interface, directly at a predetermined memory location in the main processor local memory. After a receiver acknowledges the header character (which may consist of destination ID, word count, priority and message types), additional characters are transmitted by the source as specified by the word count of each message block.

In order to transfer data from the main processor to the bus output the main processor interrupts the FEP via the processor to processor interface. The data transfer flag for each receiver is stored in the RAM of the main processor. After the FEP acknowledges the interrupt, it reads the data transfer table if there is a request for a data transfer. The FEP initiates the output of that data block to the system bus interface. Transferring input data to its main processor is accomplished in a similar manner. Since program and local variables are stored in the local memory of each main processor, interprocess communication is greatly reduced.

Input/output module and memory module

Both the I/O module and the memory module (Figure 1) have an FEP for handling data transfers to and from the bus, and have identical system bus interface, FEP, and local memory. The memory interface in the memory module performs the function of interfacing the FEP with the bulk memory. The channel interface in the I/O module asynchronously multiplexes the channels and provides asynchronous communication with its FEP, I/O control, and interrupt facilities.

The I/O module interfaces the system with the user and with a variety of peripheral devices that include various hardware interfaces. The I/O module stores the data in its own local memory which queues the data for output to an I/O channel interface. In a similar manner, the FEP retrieves data from its local memory and sends the data to the bus. The interconnection of the FEP of the I/O module to the channel interface depends on the specific application and channel interface characteristics.

PERFORMANCE EVALUATION

In examining the performance of a distributed computer system, we are interested in the study of inter-module communications, processor and bus utilization, and response time and throughput of the system. Since these performances are program and application dependent, analytical models that can accurately predict the system performance are difficult to devise. An alternative method is to construct a prototype and measure these performance values. This approach not only requires the development of hardware prototypes, software system, data reduction and measurement facilities, but also encounters difficulties in evaluation of the system performance at different parameter values. We therefore use simulation techniques which enable us to use the actual program behavior and workload profile in studying system performance.

The simulation model

In order to provide a general yet accurate simulation model for our study the model is based on events. Events are generated at each processing module according to a given arrival distribution and enter into multi-level queues waiting their turn to be processed. The processing time of each task is known. In evaluating the performance of the system we are only interested in the task processing sequence, the task time and the bus occupancy time. The information of processor instructions and bus traffic contents is not required. As a result, the model does not require the detailed knowledge necessary for the applications program and system software. This greatly simplifies the simulation. The accuracy of the result depends on the accuracy of the estimates of processing times, bus occupancy times of various tasks, and the workload profile.

The simulation model for a module is shown in Figure 3. The bus occupancy time of a task can be estimated from the number of bits transmitted and the bus bandwidth. All the queues are of a multi-level queueing system, with entry level determined by the priority of the messages. The model also has facilities to gradually increase the message priority as a function of the amount of time since it entered the system. The simulation is event-driven. That is, it advances the clock to the next event which is to occur, rather than stepping

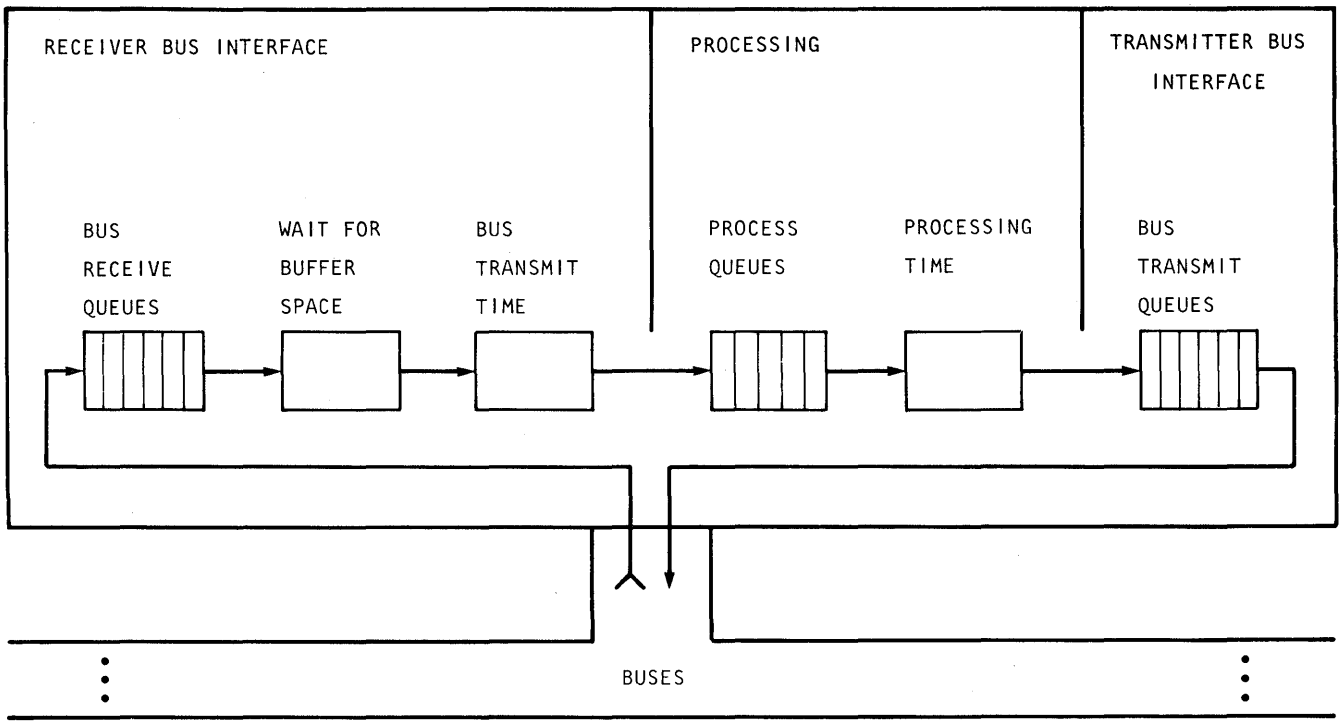


Figure 3—Simulation model for a module

through every clock interval and checking for events. Thus the time required for a simulation run depends on the number of events which occur during the simulated interval rather than the time span of the simulated interval. The internal unit of time is expressed in terms of bus clock time. Inputs and outputs are represented in milliseconds.

A module can receive data simultaneously from several busses, but can send data only on its designated send bus. The number of received busses, the number of modules in the system, as well as the buffer space can be easily varied in the simulation.

Input parameters for the simulation are message arrival rate, which corresponds to the satellite communications channel bandwidth, effective processor execution time, which reflects both instruction execution time and the power of the instruction set, bus bandwidth, assignment of processing tasks to the modules, and number of processing modules.

The performance measures used in the simulation are utilization of the processors, busses, and peripheral devices, delays at each module such as processing queues, sending queues to the bus, receiving queues from busses, and message response time.

The simulation program creates a message control block for each network message to be handled by the system (send or receive). Each block is composed of sub-blocks which specify the details of processor and bus usage and contain time stamps applied at the times of queue entry and departure. After the processing of the message has been completed, the time stamps in the message control block are used to calculate the delay statistics.

The simulation is written in PL/C, Cornell University's compile-and-go version of PL/1. Because subroutine calls have a high executed time overhead, no subroutines are used. Integer variables were used whenever possible in order to save execution time. The processing time required on the IBM 360/91 for a typical simulation run ranges from one to two minutes.

EXAMPLE FOR SIMULATION

The application selected for simulations was a communications processor (at a ground station) in a military satellite-based communications network. This is mainly because reasonable estimates of the computational requirements, workload profile, as well as program structure (program size, execution sequence and times) were available from a previous study that was implemented with a military computer (AN/UYK-20).[1,2]

The communications network shares a single half-duplex channel of a satellite in stationary earth orbit. Reservation time division multiplexing (statistical multiplexing)[3,4] is used for multiplexing the communications from all the stations. There are two types of stations: master station and slave station. The master station not only handles all the slave station tasks but also it handles time slot reservation for all the slave stations. In our investigation we shall study the performance of the communication processor in a master station. The basic time unit (time slot) is the time required to transmit a 4,164 bit block. These time slots are grouped into a frame which consists of 15 to 32 consecutive time

slots. The first time slot of each frame is for reservation of the remaining time slots in that frame. The master station makes the time slot assignments on the basis of requests received from all the other stations (slave stations). These requests, which indicate the priority and number of messages a slave station is waiting to transmit, are sent during designated assignment request time slots. An assignment permits a designated station to broadcast a message block to all other stations. The stations addressed by the header (part of the 4,164 bits) need to fully process the message. For our application, we always keep the channel busy. Therefore, dummy messages are transmitted (assigned by the master station) when there are not enough real messages to occupy all the available time slots. The amount of work involved in handling these dummy messages by the communication processor at the received station is similar to the handling of those messages that are forwarded to the other stations. Each block of data consists of a 16 bit cyclic redundancy check code. Error detection and retransmission are used for error control.

Assumptions used in the simulation example

In the simulation example, we assume the round trip propagation delay between the radio (ground station) to the satellite is 260 msec. All the peripheral devices are assumed to have constant service times: Disk=90 msec; Printer=200 msec (at 6000 lines/min.); Magnetic tape drive=150 msec. Processor execution time ratio is relative to a typical military minicomputer (AN/UYK-20) with instruction execution time of 1.5-2.3 msec. Execution time includes the effects of both instruction execution cycle time and the power of the instruction set. Each module has 16K bytes of buffer space available for bus traffic and can receive data from three busses (i.e., three other modules) simultaneously. A message fits exactly into one message block (4,160 bits including message header and cyclic redundancy check bits). Processing of a send message has a higher priority than the processing of a received message. This quarantees that send processing suffers minimum of delay, which assures that messages to be sent will be ready for transmission in their assigned time slots (as required by the network protocol) with a reasonable processing scheduling scheme. The bus bandwidth is assumed to be 50 kHz/bus.

Task partition and assignment for the example

The two main tasks of the communications processor at the master station are to perform send and receive message blocks. Their task flow sequence is shown in Figures 4 and 5. Additional tasks such as generation of network and station status reports for the operator, creation of checkpoint tapes for system recovery in case of failures, etc. occur relatively infrequently. For purposes of simplicity they are not included in the simulation.

The basic principle of a good assignment of processing tasks to processing modules is to balance the processing load and minimize the amount of inter-module communica-

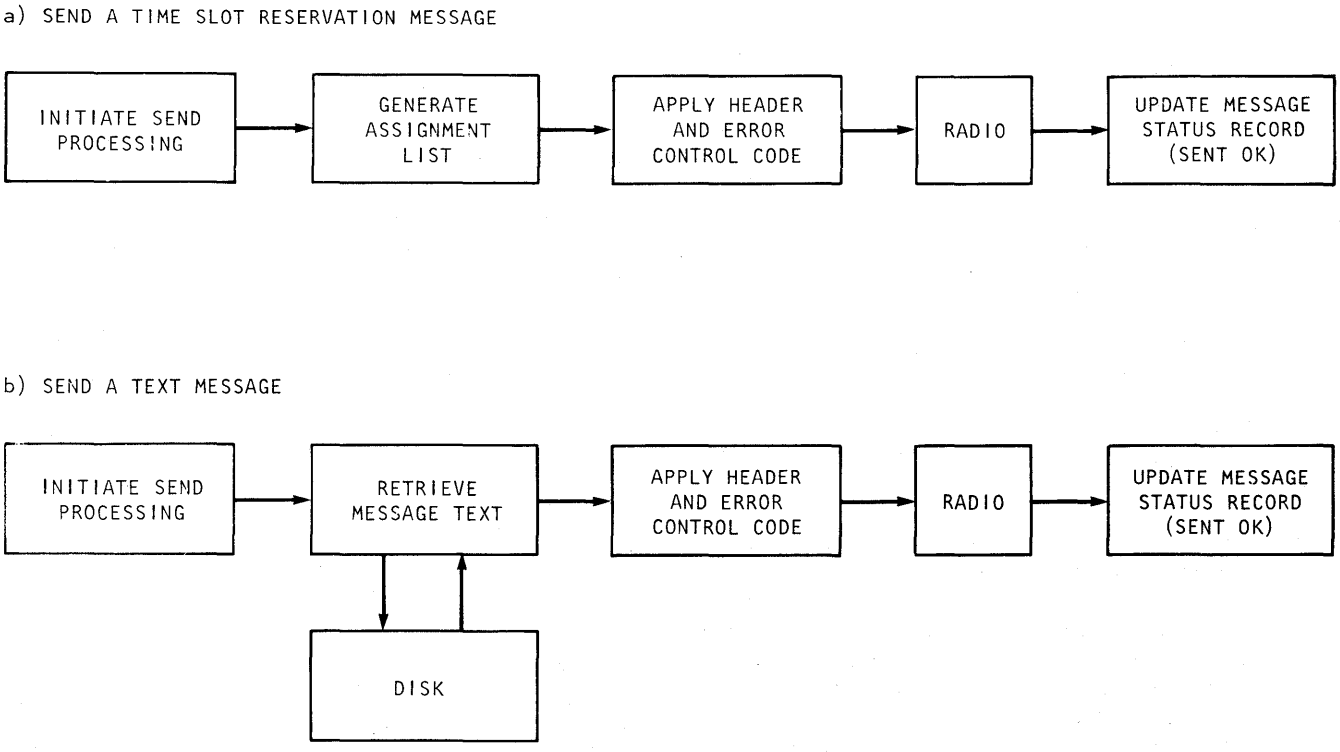


Figure 4—Sequence of processing tasks for handling a send message

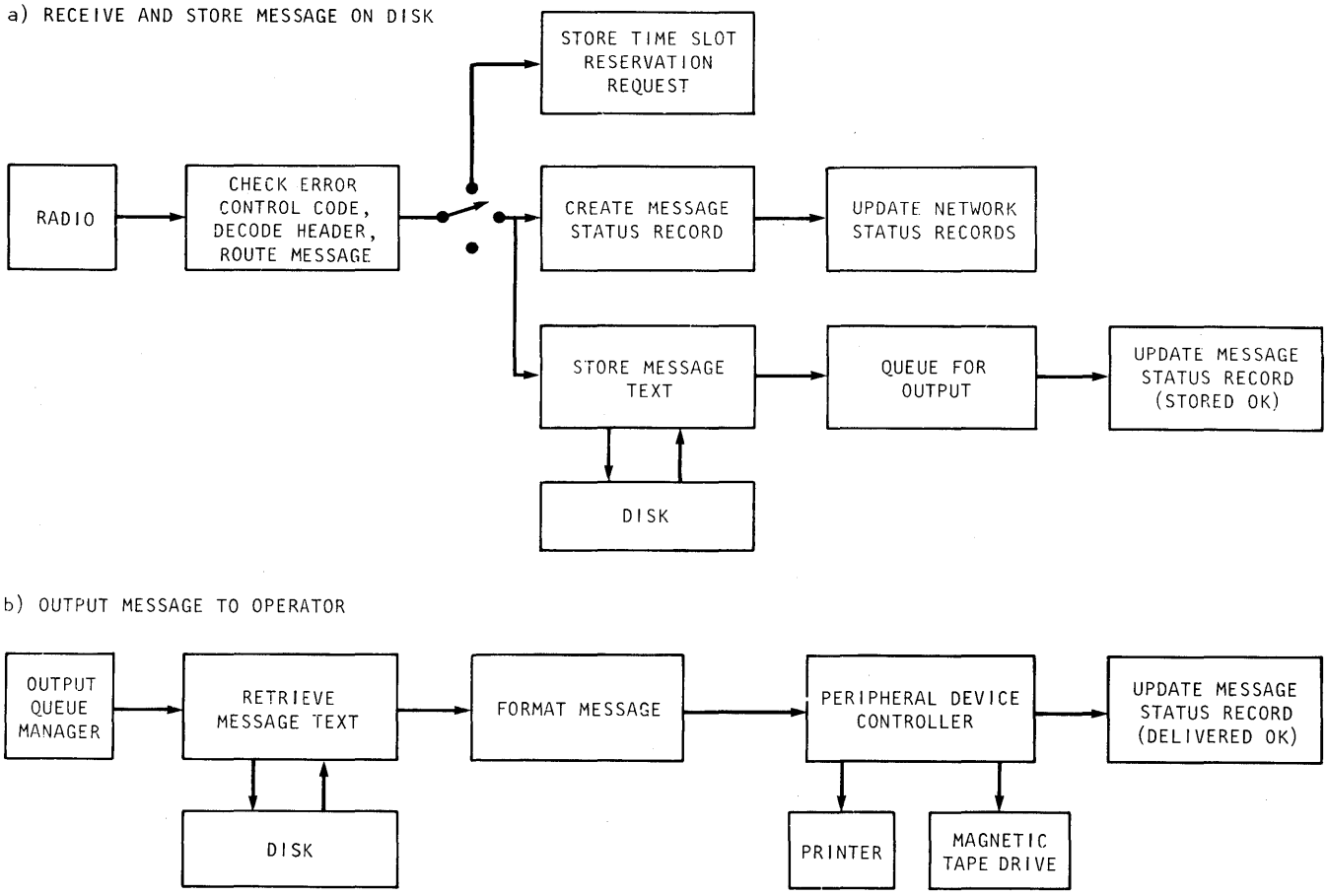


Figure 5—Sequence of processing tasks for handling a receive message

tion. Assigning all the tasks to one module requires the least amount of inter-module communication but may result in long queueing delays due to processor overload. On the other hand, assigning each task to a module yields short queueing delay for the processors but requires a large amount of time for sending intermediate results between modules. Because of the inter-module communication, processing requirements are rather complicated and may vary from one application to another. We shall use the simulation model to compare the performance of two task assignments.

The first assignment consists of assigning the tasks to a five-processor module system as shown in Figure 6. The task for sending a message on the communication network is largely independent of the task for receiving a message from the network. To minimize inter-module communication between the send task and the receive task, they are assigned to separate processor modules. We assigned one module for status record updating and time slot reservations, one module to serve as peripheral device controllers, and one module for file management and system recovery.

The second assignment consists of assigning the tasks to a three-processor module system as shown in Figure 7. Both sending and receiving processing require access to status records, but usually there are more receive messages than send messages. In order to minimize the amount of inter-module communication, status record updating and time slot reservation operations are combined with the receive processor. System recovery and report generations are combined with the send processor. The file manager and disk controller are combined, along with the peripheral device controllers, into a single module.

Table I displays the performance comparison of the three-processor and five-processor module systems. We notice

TABLE I.—Performance Comparison between the Three-Processor Module and the Five-Processor Module Systems

Number of processor modules	5	3
Virtual time to process a received message (msec)	1601	1510
Average queueing delay to receive a message (msec)	14	13
Total # of bits transmitted on buses to process one received message	13504	8736

The system is operated at the following condition: 59 percent of the received messages are addressed to the master station and require complete processing; Network data rate=9600 bits/sec.; Bus bandwidth=50kHz; processor speed equal to a minicomputer.

that the average delay in processing received messages is almost the same in both of these configurations. Due to less inter-module communication in the three-processor module system, less bus traffic is required to transmit the intermediate results and less processing time is required to prepare and interpret these inter-module communications. As a result, the three-processor module system yields better performance than that of the five-processor module system.

It is interesting to note that due to the excessive inter-module communication and the overhead in handling these

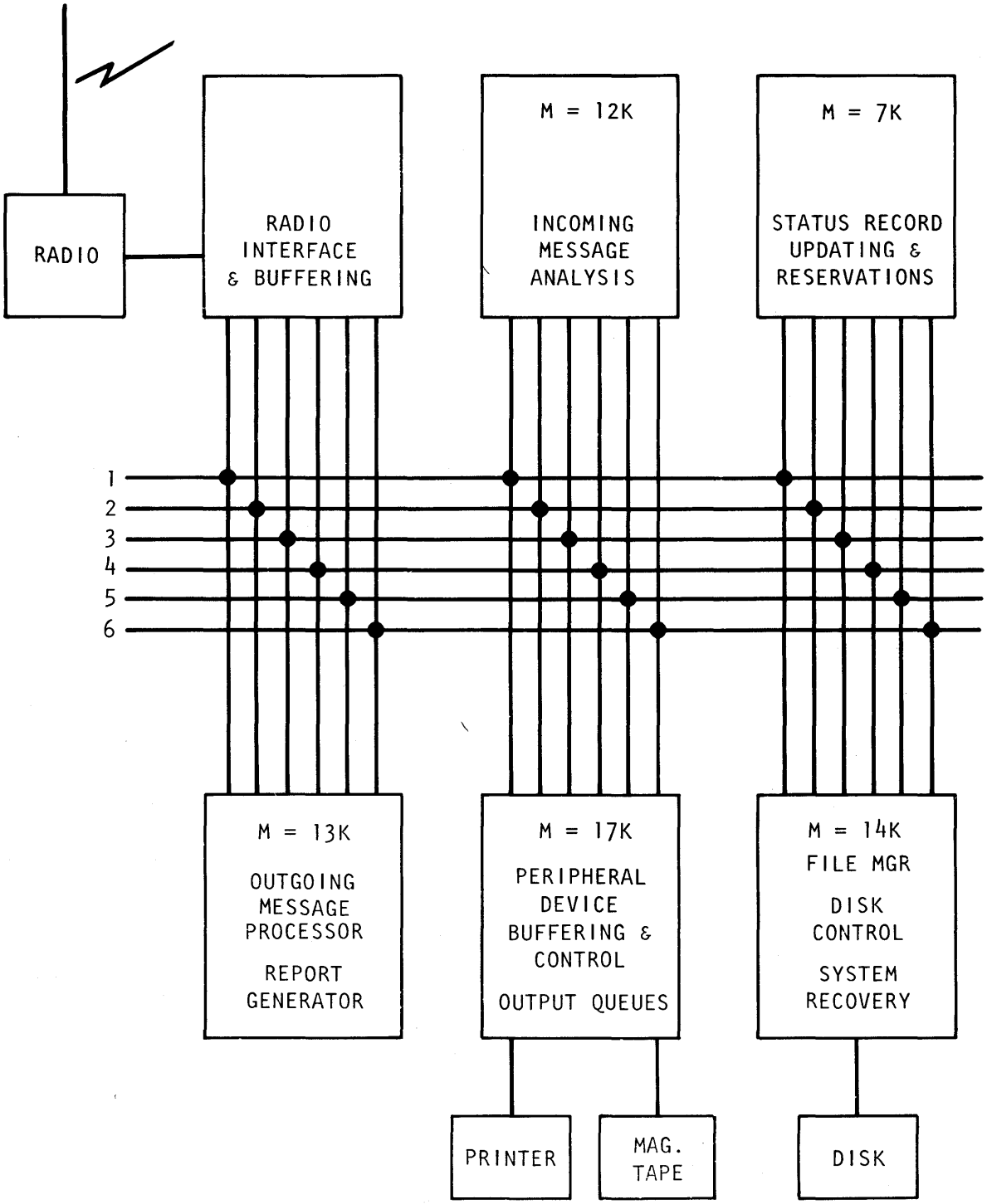


Figure 6—A five-processor module organization for the example.
M=Memory requirements for program storage (in bytes)

TABLE II.—Processor and Bus Utilization Comparison between the Three-Processor Module and the Five-Processor Module System

Utilization	Processor		Bus	
Number of processors	5	3	5	3
	Percentage		*Percentage*	
Receive processor	5	5	9	9
Status tables	2		1	
Send processor (SP)	4	4	7	7
Output queues and printer, tape controller	1	5	1	6
File manager and disk controller	5		15	

inter-module communications in the five-processor module configuration, the throughput as well as response time of the three-processor module are better than that of the five-processor module system. Therefore, in the planning of a distributed computer system, using more processors does not necessarily always yield better throughput. In fact, one of the important problems in the planning of a distributed processing system is how to partition the tasks and optimally assign them to the processor modules such that they yield minimum interprocess communications and high system throughput, and yet satisfy the cost and response time requirements.

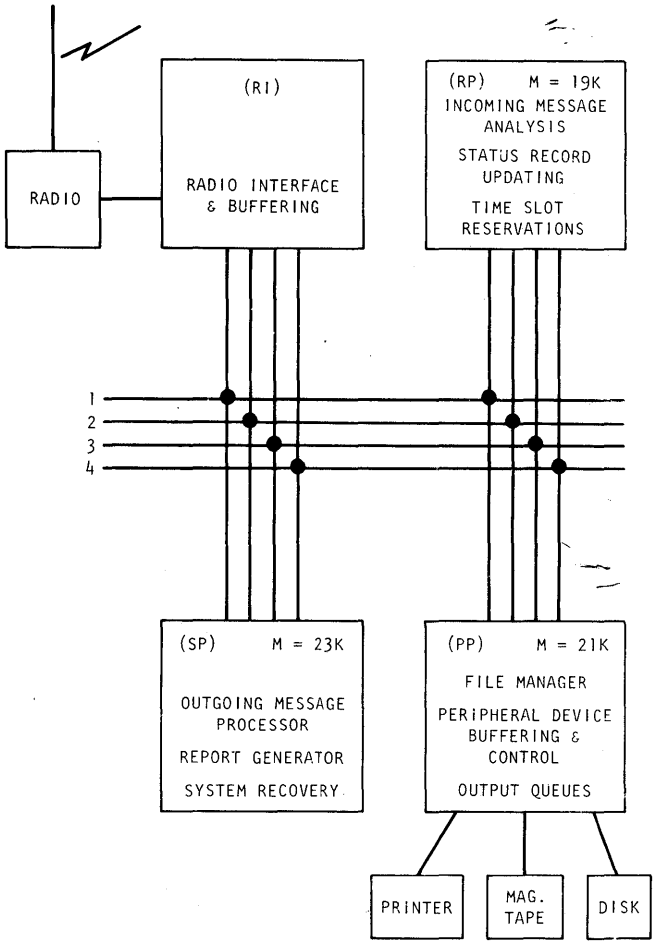


Figure 7—A three-processor module organization for the example. SP=Send processor, RP=Receive processor, RI=Radio interface. M=Memory requirement for program storage (in bytes)

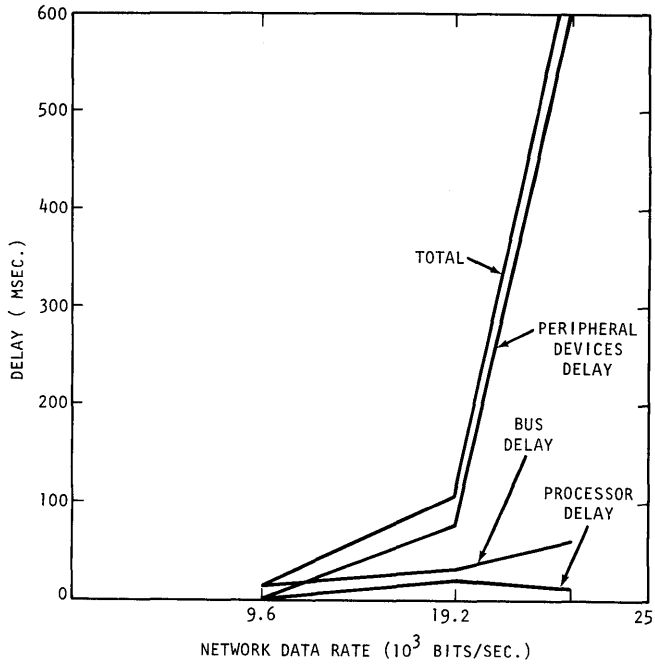


Figure 8—Average delay to process a receive message. Note that the sum of the delay components is greater than the total delay because some delays occur at the same time. Relative processor execution time=1. Bus clock period—20 msec (50 kHz/bus)

Discussion of results

Total delay is defined as the time difference for processing a message with and without any queueing delay. Total delay is not necessarily equal to the sum of all the queueing delays encountered in the processing of a message, since some processing (and delays) occur in parallel execution branches. Delays are shown only for the processing of received messages, since the processing of send messages has a higher delay allowance than that of the receive messages. Since the message arrival and departure process are deterministic and there is a long radio propagation delay, there is no interference between network cycles.* And since the processing of received messages is completed before the processing of send messages begins, we only need to simulate one complete network cycle. We studied the simulation variations of the network input data rate and relative processor execution time with the three-processor module configuration. The system performance (with processor execution time equiv-

* A network cycle consists of 32 time slots: 1 for time slot assignments by master station, 9 slots for sending messages, and 22 slots for receiving messages.

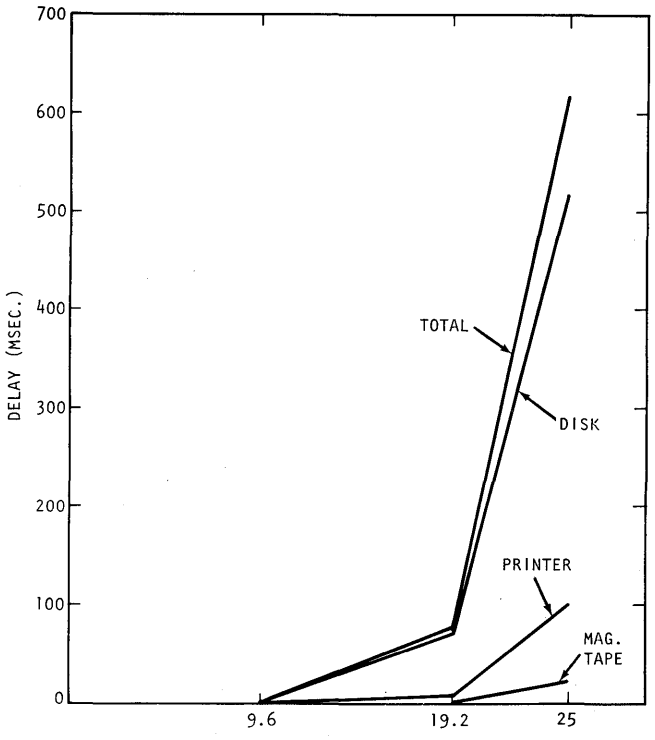


Figure 9—Peripheral device delay for processing a receive message vs. network data rate (10^3 bits/sec). Note: mag. tape drive delay is not included in total because it always occurs at the same time but is shorter than the printer delay. Relative processor execution time=1. Bus clock period=20 msec (50 kHz/bus)

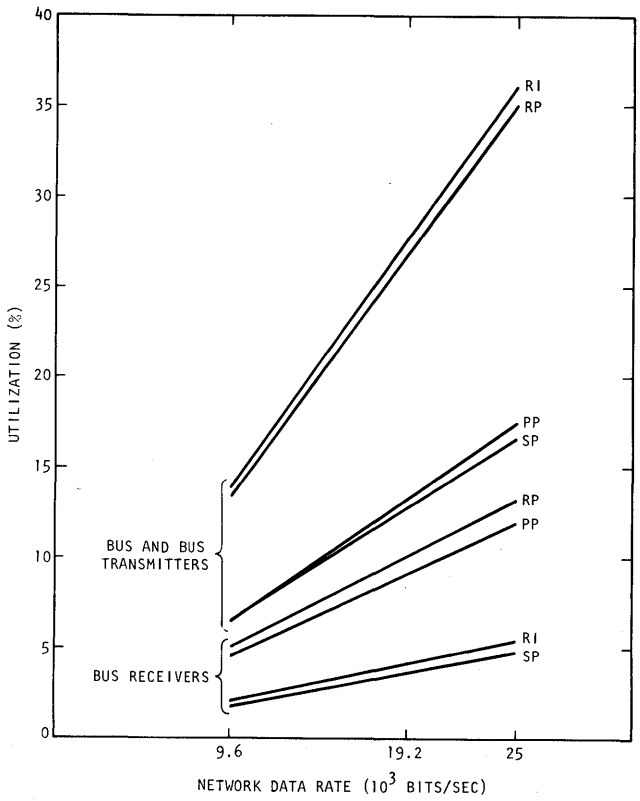


Figure 10—Utilization of busses and bus receivers vs. network input data rate. The bus transmitter utilization is the same as the bus utilization. Bus clock period=20 msec (50 kHz/bus)

alent to a minicomputer) for different network input data rate are portrayed in Figures 8, 9 and 10. From Figure 8 we notice that the largest component of total delay is due to queueing at the peripheral devices, to which the disk contributes the largest amount (Figure 9). The disk delay is large because it has a high utilization. From Figure 10 we notice that the bus utilization is only 5 percent at input rate equal to 9,600 bit/sec. Thus the bus delays (Figure 8) are small compared to an average bus occupancy time* of 44 msec. From Figure 10 we noticed that the bus receiver utilization is lower than the bus utilization because the bus receiver utilization is computed on the basis of each module being able to receive from up to 3 busses simultaneously. Since no module is addressed by more than three other modules, there is no queueing delay for bus receivers. Due to the low processor utilization (Figure 11), the processor delay is very small compared to the longest uninterrupted processing time of 38 msec for the SP module and 13 msec for the PP module. Next we studied the performance as a function of relative processor execution time. A network data rate of 19,200 bits/sec was used. The simulation results are portrayed in Figures 12, 13 and 14.

For the relative processor execution time exceeding 5 (Intel 8080 has a relative processor execution time of approximately 4 in this application), the total delay for processing a receive message exceeded the maximum allowable time which is 218 msec for 19,200 bit/sec and 436 msec for 9,600 bits/sec. From Figure 12 we noticed that for the system with large relative processor execution times, the delay of a received message is dominated by the processor delay. The increase in link queueing delay with increasing processor execution times is due to the increases in the effective peripheral device service resulting from the reduction in processor speed. The decrease in disk queueing delay at high processor execution times is due to the stretchout of

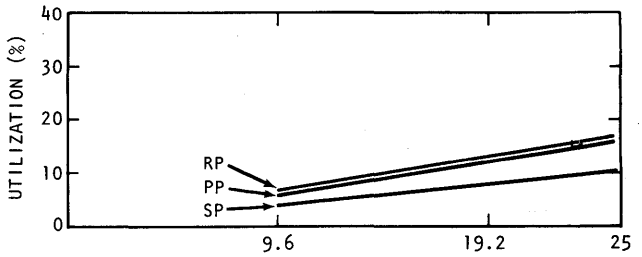


Figure 11—Processor utilization (for both send and receive messages) vs. network data rate. Relative processor execution time=1.

* Based on approximately equal numbers of large blocks of 4,192 bits each (about 85 msec) and small blocks of 192 bits each (about 4 msec) are sent on the busses during the processing of messages.

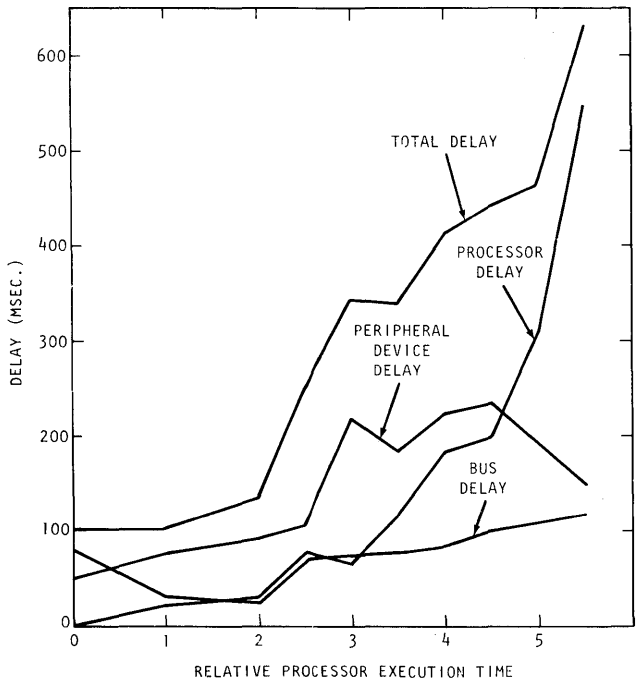


Figure 12—Average delay to process a receive message vs. relative processor execution time. Note that the sum of components delay is greater than total delay because some delays to a message occur at the same time. Network data rate=19,200 bits/sec. Bus clock period=20 msec (50 kHz/bus). The relative processor execution time for a minicomputer=1, Intel 8080 type processor=4.

access request arrivals resulting from the increased processor queueing delay. The variation in bus delay to received messages is the result of specific timing relationships between the processing of send and receive messages that occur at certain processor rates, which is due to the variations of starting time for processing of send messages. The queueing delay at the Receive Processor for processing a receive message is approximately equal to that of a PP module (i.e., balanced delay). The delay from the PP module is due to the larger number of processing tasks, each of shorter duration that for the SP module. Processor utilization (Figure 14) is the overall utilization average for the whole network cycle. When the processor execution time is large, the demand for processing may be higher than 100 percent during part of the time. This yields a very large queueing delay.

With the three-processor module configuration with Intel 8080 type microprocessors, the system performance is adequate for handling of the example network with a data input rate of 9,600 bits/sec. Since three-processor modules are just adequate, it is very unlikely that a system with a two-processor module configuration could do the job satisfactorily. The response time for the two-processor module is likely to be much larger because the processing tasks cannot be easily divided into two equal parts. Consequently, additional delays will be caused by increases in contention between processors and by unbalanced processor loading.

Simulation results reveal that the serial broadcasting bus structure is viable. The bus delays for receive messages are lower than the processor delays for a relative processor execution time greater than 3 and are comparable to the processor delays for a relative execution time in the ranges between 1 to 3 (Figure 12). Bus delay can be reduced con-

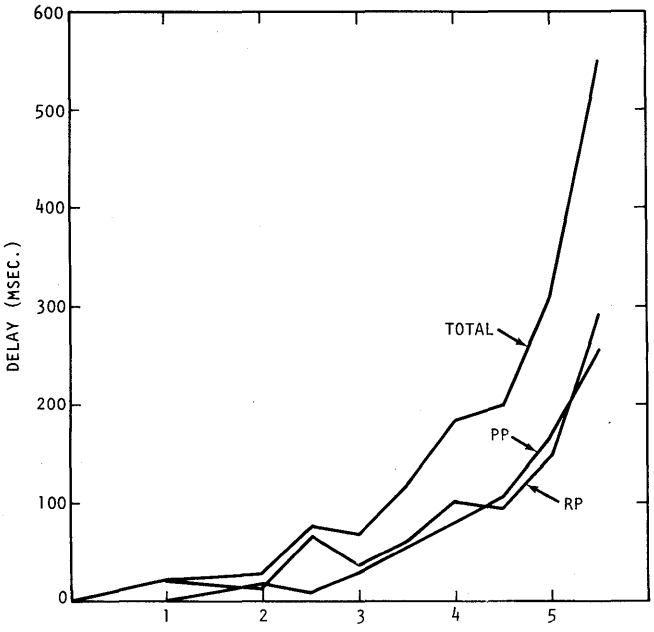


Figure 13—Average delay at the processor module for processing a receive message vs. relative processor execution time. Network data rate=19,200 bits/sec. Bus clock period=20 msec (50 kHz/bus)

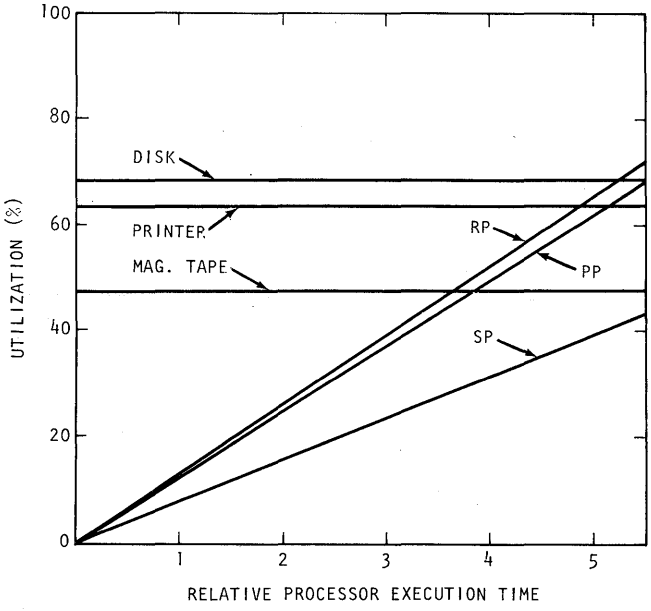


Figure 14—Processor and peripheral devices utilization (includes processing of both send and receive messages) vs. relative processor execution time. Network data rate—19,200 bits/sec.

siderably with a higher bandwidth bus. A much higher bus bandwidth interface with 2.5 MHz (50 times higher than the bus bandwidth used in the simulation) is available commercially.

We also noticed that increasing the network data rate causes a higher bus and processor utilization, while increasing relative processor execution times increases only the processor utilization.

CONCLUSIONS

Simulation study revealed that the proposed distributed processing architecture with three processor modules provides adequate processing capability for handling the Naval data communication network processing tasks. Further, the serial broadcasting bus structure provides sufficient bus bandwidth for the application.

We also found that task partition and task assignment to various processor modules has great influence on interprocess or intermodule communications and thus affect the system throughput. A good rule for optimal task partition and assignment is to balance the workload among the various modules and yet keep them as independent as possible so as to reduce inter-module communication. Excess intermodule communication not only requires excess bus bandwidth but also requires extra processing time for handling these inter-module communications. This degrades system throughput.

The low bus and processor utilization for the system indicates that our proposed distributed processing architecture would be useful for other applications as well. Distributed processing architecture not only is more flexible for system reconfiguration and system expansion, but it also provides greater fault tolerant capability. Because of these capabilities and its low implementation cost, distributed processing architecture should find its place in many future applications.

ACKNOWLEDGMENT

The authors wish to thank Mr. Howard Wong and Ms Dana Small of Naval Oceanic System Center, San Diego, and Joel Trimble of the Office of Naval Research, for their encouragement and support in carrying out this joint project between NOSC and UCLA.

REFERENCES

1. Kelly, R. and B. Iffla, "Programmable Link Adapter Subsystem (PLAS) Design Plan," Naval Oceanic System Center, San Diego, California, TN2675, 20 May 1974.
2. Knight, T. A. and C. C. Cooke, "Computer Program Design Specification for Common User Digital Information Exchange System (Shore Subsystem)," Naval Oceanic System Center, San Diego, California, TN2717, 24 June 1974.
3. W. W. Chu, "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computer Systems," *AFIPS Conference Proceedings*, Vol. 35, 1969, pp. 669-678. Also in *Advances in Computer Communications*, W. W. Chu editor), Artech House, 2nd ed. 1976.
4. L. G. Roberts, "Dynamic Allocation of Satellite Capacity through Packet Reservation," *AFIPS Conference Proceedings*, Vol. 42, 1973, pp. 711-716. Also in *Advances in Computer Communications*, W. W. Chu editor, Artech House, 2nd ed. 1976.

Integrated optimization of distributed processing networks

by W. CHOU
North Carolina State University
Raleigh, North Carolina

and

F. FERRANTE and M. BALAGANGADHAR
Page Communications Engineers, Inc.
Vienna, Virginia

INTRODUCTION

During the next decade, in the wave of microelectronics and digital revolution, we can expect to witness fast growth in the number of communication-based distributed data processing systems. A communication-based distributed data processing system is a decentralized data processing system in which data processing facilities or computer resources are geographically distributed and communicate with each other through a data/computer communications network. Figure 1 typifies such an environment. The most common data/computer communication network architectures used in the distributed environment are those illustrated in Figures 2, 3 and 4. These are the tree-shaped hierarchical, the packet and the ring-shaped architectures, respectively. These architectures are not new and heuristic algorithms have been widely available for their optimization. However, the emphasis on the distributed data processing systems shifts the application environment and objectives of the optimization problem and hence lessens the applicability of these earlier algorithms. This paper presents a set of integrated algorithms more appropriate for network optimization in this new environment. Some of the algorithms are newly developed; some are modifications of the earlier ones.

Figure 2 depicts the tree-shaped, hierarchical architecture, in which the local access network defined as the circuits interconnecting the terminals and/or terminal control units, are connected to, and controlled by, the concentrating devices. The concentrating devices are in turn connected to, and controlled by, the host computer. The concentrating functions may be performed as part of a remote processor's responsibility. The circuits shown interconnecting the concentrating devices and the host computers are called the backbone network. Both the local access and the backbone networks may be point-to-point or tree-shaped connections. Occasionally, ring-shaped terminal connections may also occur.

The mesh-shaped packet switching architecture is illustrated in Figure 3. In this architecture the local access network comprises both the terminal devices and the host computers connected in either a point-to-point or multipoint configuration. The local access networks are connected to the packet switches. The backbone network for this architecture is defined as the interconnecting circuits between the packet switches.

Figure 4 presents the architecture associated with a ring-shaped, ring-switching design. The local access network in this case is the same as that defined earlier for the packet network architecture. Similarly, the backbone network is defined by the circuits interconnecting the switches, in this case specified as ring switches. For completeness, it should be noted at this point that terminals controlled by a Terminal Control Unit (TCU) usually form, on a second order local level, a star or ring network as shown in Figure 5.

As evident from the figures, interfacing the local access and backbone networks are the concentrating devices, the packet switches, and the ring switches. In this paper, these devices will be generically referred to as the Network Access Facilities, or NAF's. (For more detailed descriptions of the data network architectures, see Reference 1.)

The network optimization problem of a distributed system, therefore, consists of the determination of the number and locations for NAF's; the configuration of the local access networks, including point-to-point, tree-shaped multipoint and ring-shaped multipoint; and the configuration of the backbone network, including point-to-point, tree-shaped multipoint, ring-shaped multipoint, and mesh-shaped packet switching architectures. Of course, the optimization of each of these network problems is not new, and good heuristic algorithms have been widely used in many data/computer communication networks. However, new emphasis on the distributed data processing systems has lessened the applicability of these algorithms. The following sections of this paper discuss the algorithms and implementation approaches that are felt to be more suitable to the new distributed processing network environment.

For local access networks with geographically dispersed terminal devices, "effective" algorithms for optimization have existed for over a decade.[2,3] The term "effective" implies nearness to the theoretic optimum. With the number

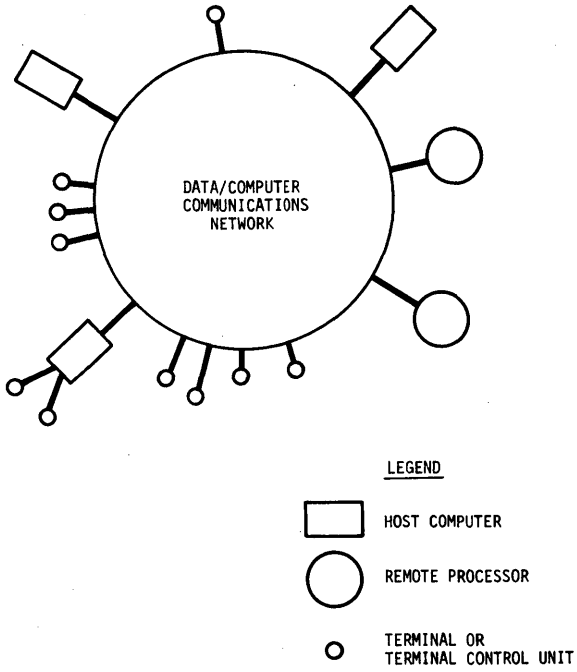


Figure 1—Communication based distributed data processing system

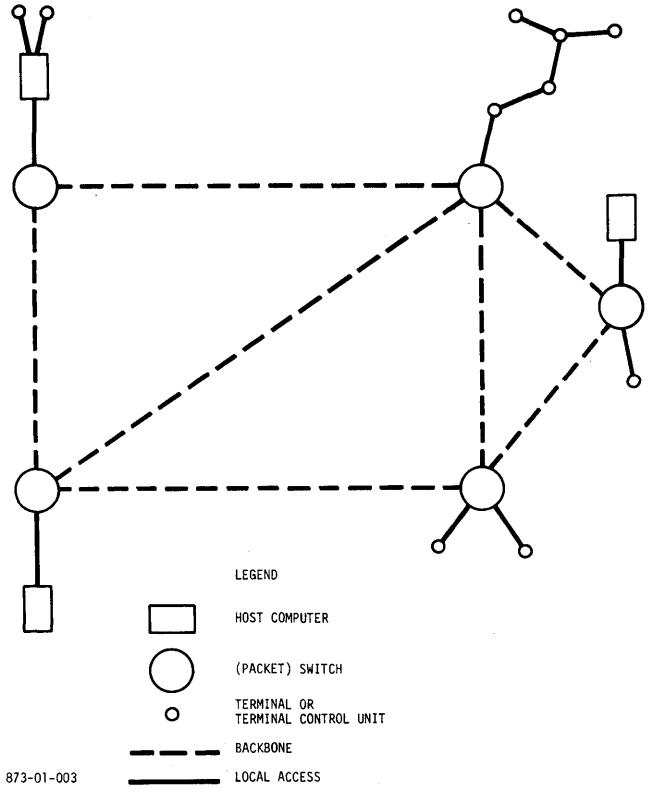


Figure 3—A packet switching architecture

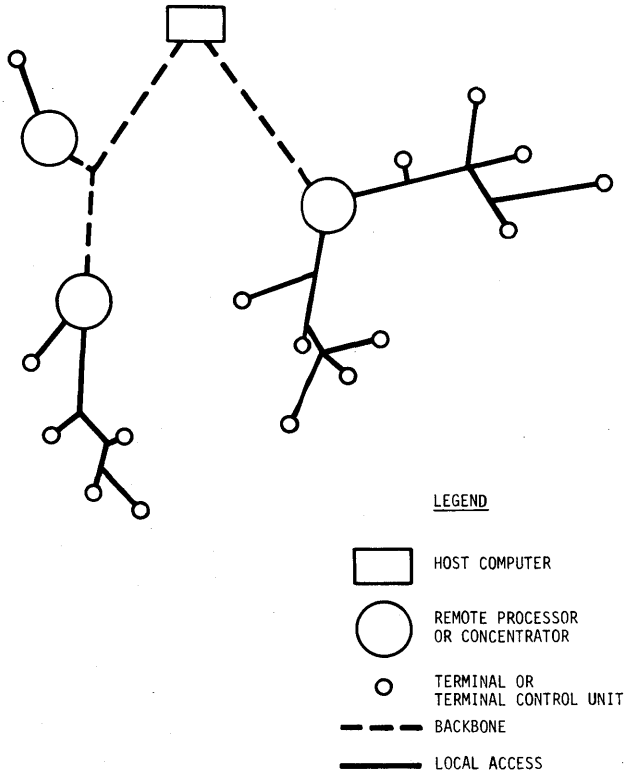


Figure 2—Tree-shaped hierarchically controlled data communication network architecture

of terminals on the order of hundreds or even thousands, the efficiency in terms of computer memory and running time requirements for executing the algorithms becomes important. Otherwise, however effective, implementation of the algorithms may become too expensive. The second section describes a currently most effective algorithm for the optimization of tree-shaped multipoint networks and shows how it can be modified to improve its implementation efficiency.

The concept of ring-switching has been proved workable for quite a few years,[4] and there are ring-shaped local access networks in operation; e.g., IBM 3600. However, with few exceptions, devices on a same ring are generally located in the same proximity. Therefore, topological optimization has never been a concern and there has not been direct published work in this area. Nevertheless, it is expected in the distributed data processing environment that in some applications, the ring network may be more desirable than the tree network even for geographically dispersed devices. The third section describes algorithms for designing ring-shaped networks, based on the modifications of graph-theoretic results.[5,6]

Dominating the backbone network optimization problem is that of the packet switching network. The bulk of the research in the optimization of packet switching networks has been supported by the Advanced Research Project Agency from 1969 through 1976. Naturally, the algorithm that has been widely used and has been verified by real

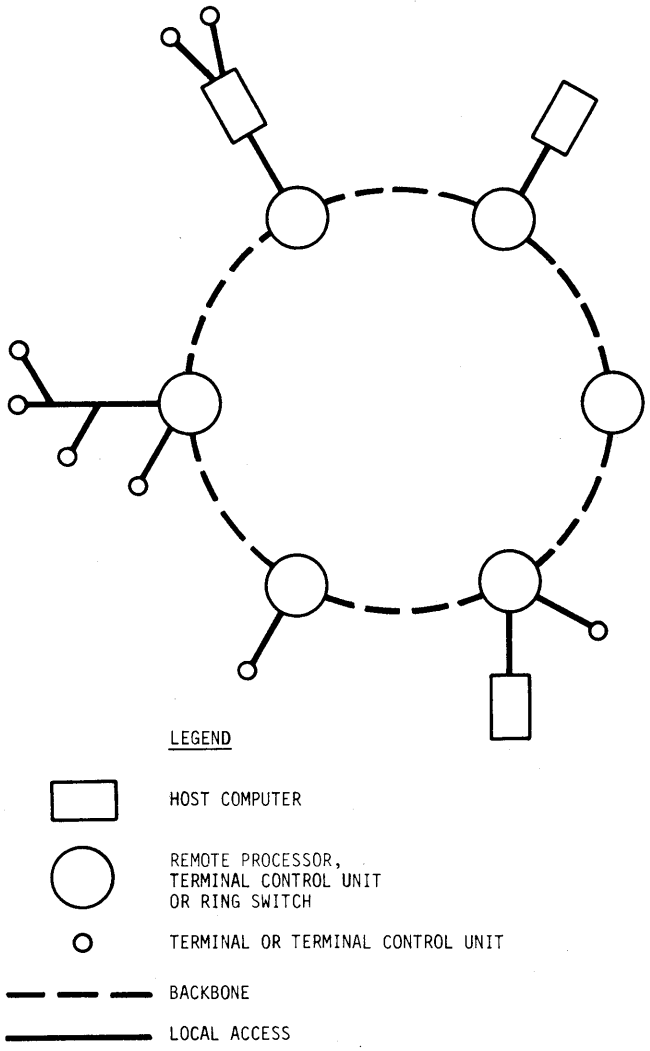
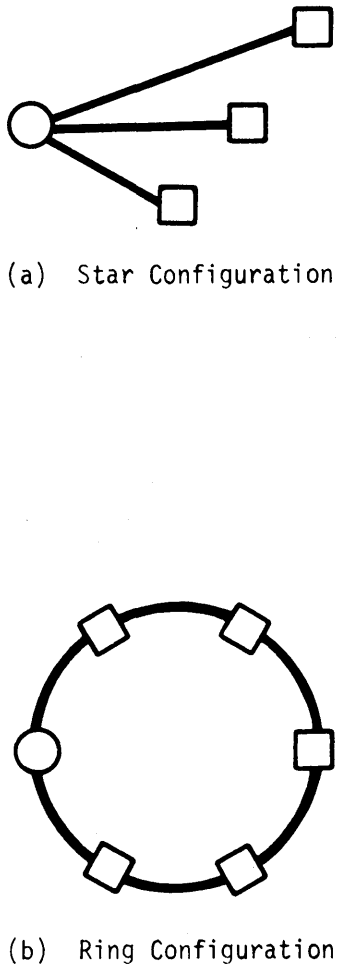


Figure 4—Ring architecture

networks is the one sponsored by the ARPA, and just as naturally, these algorithms have been biased by the ARPANET environment during that period. The fourth section describes these biases in the algorithm, and specifies a modified algorithm that eliminates these biases and improves both their effectiveness and efficiency.[7]

In the past, the number and locations of the NAF's have not been an important consideration for optimization. They are more constrained by users' organizational structure than costs. As a consequence, algorithms developed to date are more of an academic interest than for real network optimization. Hence, definite room exists for the improvement in these algorithms' effectiveness and related efficiency. In particular, even though the costs of the local access and backbone networks depend on the number and locations of the NAF's, these algorithms have not sufficiently considered these configurations in making decisions on the number and locations of the NAF's. The fifth section develops an effective and efficient algorithm for the determination of the number and locations of the NAF's.



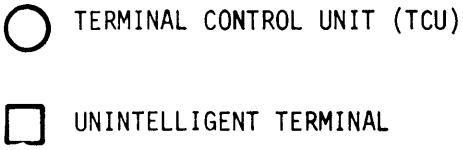


Figure 5—TCU-terminal configurations

Using conventional approaches to design a distributed network as shown in Figures 2, 3, or 4, the number and locations of the NAF's, the configuration of the local access network, and the configuration of the backbone network are each treated as a separate optimization problem. The optimal network cannot be realized by simply overlaying three optimized subnetworks. To avoid such a suboptimum design, it is necessary to integrate consideration of the subnetworks in the final and overall optimization process. The fifth section, in conjunction with the determination of the number and locations of NAF's, introduces an integrated approach that combines the three optimization problems

into one. This integrated approach, together with the algorithms given in earlier sections, has been implemented as a computer program. It is capable of designing in one single execution, a distributed network of the form shown in Figures 2, 3, or 4. In addition to the integration of three optimization problems into one process, the program which has been developed also integrates into the same process the different network shapes (tree, ring, mesh) and variations associated with differences in line circuit tariffs.

LOCAL ACCESS NETWORK OPTIMIZATION—TREE-SHAPED MULTIPOINT CONFIGURATION

The basic problem of tree shaped multi-point line design can be stated as: given a geographic distribution of terminals, locations of roots (network access facilities, host computers), line speeds or capacities and constraints (such as maximum number of terminals permitted per line, maximum traffic permitted per line as a function of the terminals on the line), design a least-cost network connecting the terminals to the roots.

Many, mainly heuristic, techniques have appeared in the literature for the optimization of tree-shaped networks in data communication systems.[2,3,8–11] The techniques, though differing, are all constrained minimum spanning tree algorithms. They design minimum cost tree-shaped networks satisfying the constraints. Each of these algorithms starts with the terminals in separate components and subsequently joins pairs of components. They differ only in the order in which they consider joining components. If no constraints are imposed, they all yield a least-cost tree-structured network. Since the grouping of a given set of terminals into one component (i.e., placing them on the same multidrop line) restricts subsequent merging with other components because of the constraints, these algorithms, in general, yield different solutions. For example, Prim's algorithm[9] chooses a terminal whose distance to any terminal already in the tree is minimal as a basis for joining components to the tree. Kruskal's algorithm[8] evaluates a least-cost line connecting a pair of components and joins the components by that line. (See Reference 3 for more details.)

Based on the above principle, Chou and Kershenbaum[3] have developed a unified algorithm which, with the proper choice of parameters built into the algorithm, allows one to emulate a specific algorithm and to take advantage of the precepts forwarded by many of the existing algorithms. The network designer, using the Chou-Kershenbaum algorithm, can take advantage of features offered by other algorithms or experiment with new algorithms by simply varying parameters. In addition, while most techniques allow only one root for all the tree networks during the execution of the algorithm and, in the case of multiple roots or NAF's, require a preprocessing step of clustering terminals for each of the roots or NAF's, the Chou-Kershenbaum algorithm allows multiple NAF sites to be considered during the execution of their algorithm. A modified version of the algorithm is given below.

CHOU-KERSHENBAUM'S UNIFIED ALGORITHM FOR TREE NETWORK OPTIMIZATION

STEP 0 INITIALIZATION

$A \triangleq \{1,2, \ldots, N_t\}$	(Set of all yet unconnected terminals, N_t=Number of terminals)
$B \triangleq \{N_t+1, \ldots, N_t+N_{NAF}\}$	(Set of all concentrators, N_{NAF}=Number of concentrators)
$C_i \triangleq \{i\} \forall\, i \epsilon A \cup B$	(Initially, each component C_i contains one node)
Define W_i	(Weighting function associated with C_i. e.g., min. connecting cost from C_i to any concentrator)
$t_{ij}=W_i-d_{ij},\ i \epsilon A,\ j \epsilon A \cup B$	(Cost trade-off function of connecting C_i and C_j, d_{ij}=cost of connecting i and j)
$N_L=0$	(Number of connected links=0)

STEP 1 DEFINE MAX COST TRADE-OFF FUNCTION

$t_{i^*j^*} \triangleq \text{Max } t_{ij}$	
$i \epsilon C_i \cap A$	(i must be a terminal)
$j \epsilon C_j$	(j may be either)
$C_i \neq C_j$	(i and j not in same set)

STEP 2 ADD LINKS

If $C_{i^*} \cup C_{j^*}$ does not violate constraints,

Add (i^*, j^*)	(Adding a link)
Let $C_{j^*} \leftarrow C_{i^*} \cup C_{j^*}$	(Merging two components)
$N_L \leftarrow N_L+1$	(Increment N_L)
Update W_i and t_{ij},	(W_i may vary with configurations)
Otherwise, let $t_{i^*j^*}=\infty$	((i^*, j^*) are not allowed)

STEP 3 CHECK NUMBER OF CONNECTED LINKS

If $N_L<N_t$, go to Step 1	(End of the algorithm if the number of links equals to the number of terminals)
Otherwise, stop.	

The algorithm does not restrict the definition of the terminal weight, W_i. However, Chou and Kershenbaum[3] suggests the following definition:

$$W_i = a(b \times d_{i0} + (1-b) \times d_{i2})$$

Where,

d_{i0}=Cost of connecting terminal i to the closest NAF
d_{i2}=Cost of connecting i to the second nearest neighbor
a,b are parameters of the algorithm such that $a>0$, $0 \leq b \leq 1$

The parameters, a and b, when varied allow the Chou-Kershenbaum algorithm to essentially perform operations similar to other algorithmic processes. For example, by setting $a=0$ Kruskal's algorithm is simulated; or by setting $a=b=1$ Esau-William's algorithm results.

Before, the publication of References 3 and 12 proved that most prior published heuristic algorithms achieve near-theoretic-optimum results, particularly so of the Esau-William algorithm.[2] Cain[13] then showed that Chou-Kershenbaum algorithm always outperforms Esau-William algorithm. Thus, the effectiveness of the algorithm is well recognized. However, the efficiency of the algorithm, i.e., the computer memory size and running time required to execute the algorithm, has not been shown to be optimized.

In a large network with many terminals, large memory requirement and long execution times are primarily the result of determination and updating of t_{ij} values. Technically, determination of minimum t_{ij} implies the computation and storage of (t_{ij}) for all i and j. With 1,000 terminals, this means that almost 1,000,000 lines have to be considered for costing, at each stage of the algorithm. The memory and computation time requirements can be expressed as, for large N, $K_m N^{a_m}$ and $K_t N^{a_t}$, where K_m, a_m, K_t, a_t are constants, and N is the number of terminals. With a straightforward implementation, a_m is 2, a_t is 2 or 3.[14] The computation time could be horrendous and the algorithm becomes impractical to use even if N is in the low hundreds.

In order to improve the efficiency computation time, we must lower the value of a_m and a_t. It can be observed that by proper preprocessing, the value of K_t is increased with companion consequence of lowering the value of a_t. With the following two schemes, it can be shown that we are able to reduce a_m and a_t close to "1". For large practical networks, a_m and a_t can actually be reduced to a value less than "1." (The net result is that, memory storage size and computation time are increased by the schemes for small N due to the increase in K_t, K_m; but are reduced for large N.)

SCHEME A—ONLY TERMINALS WITHIN A NEIGHBORHOOD BEING CONSIDERED FOR CONNECTIONS

Without significant degradation in the resulting solution the evaluation can be reduced to consideration of only a few t_{ij}, based on the premise that in the optimum design network, it is very unlikely that a terminal will be connected directly to a very distant terminal.

Significant execution time savings can be obtained in determining the nearest neighbors if, instead of evaluating all inter-terminal costs, the area containing the terminals is partitioned into grids. Then, for each terminal, the nearest neighbors are found by considering only the terminals contained either within the same grid or in the next adjacent grid area.

The evaluation and updating of t_{ij} can be speeded up by maintaining the neighbors as a list-data structure with associated nearest neighbor trade-off parameters. This structured list is referred to as a "heap." Updating a "heap" structured array can be shown to be much more efficient than re-sorting an identically sized array. This is due in part because the least element is always located at the top of the heap. Maintaining the neighbors in a linked list has the advantage that after merging link (i) with its immediate neighbor link (j), by simply adjusting appropriate pointers, immediate neighbors of link (j) can be assigned as link (i)'s immediate neighbor. Note, once link (j) has been merged with link (i), it is no longer necessary to consider the link (i, j).

This scheme has been implemented with FORTRAN in an IBM 370/158. Figure 6 illustrates the relationship between computation time and the number of different terminal locations. As shown, the curve is practically linear. Superimposed on the Figure is the predicted curve that would have occurred had this scheme not been employed. Let K_t, for the curve representing the implementation with Scheme A be K_t', and for the curve representing the implementation without Scheme A, be K_t''. We *conservatively* assume $K_t'=50K_t''$, and, *optimistically* assume $a_t=2$ for the curve without Scheme A. (K_t' is found to be 0.75).

With Scheme A, the computation time can be expressed as $C_0+C_1 \log N+K_T' N$. For large N, the term $C_0+C_1 \log N$ is insignificant.

SCHEME B—PREPROCESSING TERMINALS IN A SAME LOCALITY

When terminals served by the same exchange are interconnected often no mileage charges are levied. Thus they can form components and be interconnected based only on traffic constraints. By only increasing the number of terminals in each locality without increasing the number of localities, the value of "N" in $K_t N^{a_t}$ is hardly increased and the computation time is only increased slightly. For systems with more than a couple of hundred terminals, it is likely that many locations have multiple terminals. The multiplicity of terminals per location increases as the total number of terminals increases. Thus for systems with several hundred terminals or more, the computation time goes up much less than linear as the terminal size goes up. Figure 7 illustrates this relationship.

LOCAL ACCESS NETWORK OPTIMIZATION—RING NETWORKS

At the present, only a very small percentage of data and computer communication networks are ring-shaped. For those that are, the nodes on the ring network are concen-

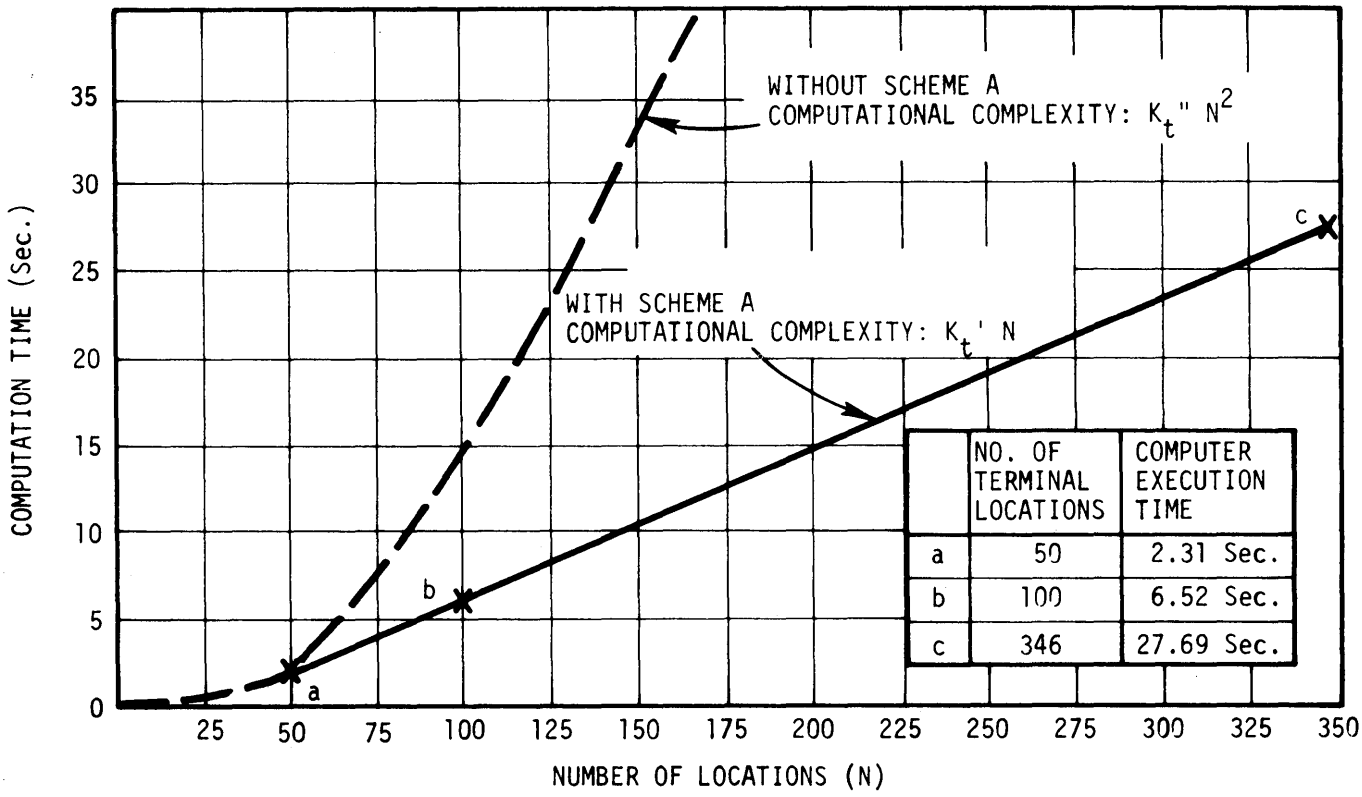


Figure 6—Computation complexity of Chou-Kershenbaum algorithm with Scheme A

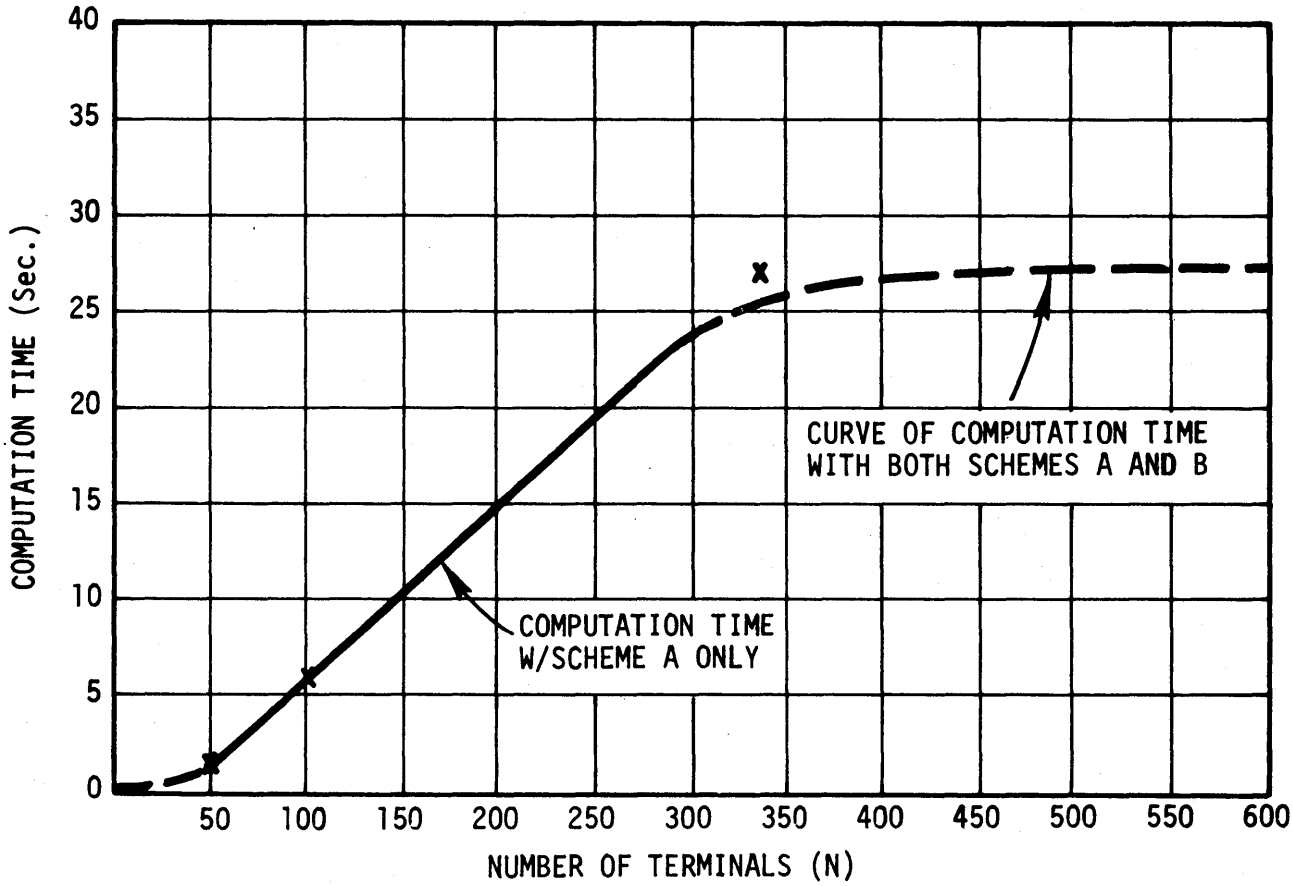


Figure 7—Computation complexity of Chou-Kershenbaum algorithm with Schemes A and B

trated within a building complex, or the number of nodes is small. Eyeball design of such networks has been adequate. And no algorithm has been published for the design of ring-shaped computer communication networks. To consider ring-shaped networks covering larger geographical areas, good algorithms are needed. Good algorithms do exist; however, they were not inspired by data and computer communication networks, but are directly applicable to the layout of ring-shaped data communication networks.

The topological layout problem for the ring-shaped data networks can be classified into two problems. One problem is to connect terminals or communication devices into a ring (loop) shaped network with minimum total transmission line cost, given the locations of the devices. This problem is equivalent to the traveling salesman problem of determining a minimum cost tour for a salesman visiting every city exactly once. It can be effectively solved by following steps: start from an arbitrary initial ring interconnecting the nodes; interchange one or more pairs of links while keeping the network in the ring-shape; if the exchange does not result in a net cost saving, restore the original network; otherwise, use the new network and repeat the interchange process until no appreciable improvement can be made.[5]

A second problem involves the layout of multiple ring-shaped networks. In this case, the number of terminal devices that can share a loop is often constrained by the terminal generated traffic and by the link capacities. This second problem is the constrained loop problem, and can be considered an extension of the traveling salesman problem. Instead of having one salesman visiting all the cities, there are now several salesmen visiting different cities. Each salesman must start from the central office. The number of salesmen and the number of cities to be visited by a salesman depends upon the sales load at the cities. The problem can be solved effectively by partitioning the terminals into separate rings and then optimizing each of these rings as a separate traveling salesman problem.

A very good algorithm for partitioning the terminals is the following one given by Clark and Wright.[6] Initially, consider every terminal as forming a separate ring connected to the central node (communication device or host computer) by two links. Merge the two rings which result in the largest cost saving without violating constraints. Each time two rings are merged, two of the four links connecting them to the central node are removed and one link connecting the two individual rings is added. This process continues until no more rings can be merged without violating the constraints. This Clark-Wright algorithm gives good results for the terminal layout design problem even without the link exchange reoptimization of the individual rings. It is also interesting to note that their algorithm is quite similar to the one for tree network design. The Clark-Wright algorithm is presented below in a form similar to tree network design algorithms.

MODIFIED CLARK-WRIGHT ALGORITHM FOR RING NETWORKS

STEP 0 INITIALIZATION

$A=\{0,1,2,3,\ldots,N_t\}$ (Set of all nodes, N_t=Number of terminals)

$C_i=\{i\}\quad \forall i\epsilon A$ (Initially, each component C_i contains one node)

$i_1=i_2=i\quad \forall i\epsilon A$ (i_1 and i_2 denote the two end points of the chain formed by C_i)

$d_{i_1}=d_{i_2}=d_i\quad \forall i\epsilon A$ (d_k=the cost connecting k to NAF)

$t_{ij}=\underset{k=1,2,\ h=1,2}{\text{MAX}}\{d_{i_k}+d_{j_h}-d_{i_k j_h}\}\forall i,j,\epsilon A$ (Det. the cost trade-off functions of merging C_i, C_j)

$N_L=0$ (Number of links=0)

STEP 1 DET. THE MAX COST TRADE-OFF FUNCTION

$t_{i^*,j^*}\triangleq \text{Maxt}_{i,j}\quad i,j\epsilon A$

STEP 2 MERGING TWO LOOPS BY ADDING A NEW LINK

If $C_{i^*}\cup C_{j^*}$ does not violate constraints, (e.g., no more than certain number of terminals per ring)

Add (i^*, j^*)

$C_{j^*}\leftarrow C_{i^*}\cup C_{j^*}$

$A\leftarrow A-\{i^*\}$

Det. j_1^*, j_2^*, $d_{j_1^*}$, $d_{j_2^*}$

Update t_{i,j^*}, or $t_{j^*,i}$ for $i\neq j^*\epsilon A$

$N_L=N_L+1$

Otherwise, $t_{i^*,j^*}=-\infty$

STEP 3 If $N_L<N_t$ Go to Step 1

Otherwise, add a link from 0, the NAF, to the free end point of $C_i\forall i\epsilon A$,

STOP

BACKBONE NETWORK OPTIMIZATION—PACKET SWITCHING NETWORKS

A backbone network can be point-to-point, tree-shaped, ring-switched, or packet switched. For the first three types techniques and algorithms given in earlier sections apply. This section discusses the optimization of packet switching networks.

The topological design of distributed packet switching network was first discussed by Frank, Frisch, and Chou.[15] They described the application of branch exchange techniques in conjunction with branch deletion and addition techniques to the optimization of ARPANET. A summary of experience with branch exchange techniques for ARPANET optimization is given in Reference 16. Similar techniques are discussed in Reference 17. Basically, these techniques seek repetitive cost or throughput improvement by adding, deleting, or simultaneously adding and deleting links, (branches, lines) to a starting network until there are no modifications that result in an appreciable improvement. An evolution from the branch addition, deletion, and exchange is the cut-saturation algorithm[7] which improves both the effectiveness of the design and the efficiency of the computer running time. A cut is any set of lines whose removal will disconnect the network. A cut is saturated if the traffic load of every link in the cut is at capacity. There are, typically, a large number of cuts in a network. When traffic load grows, one of the cuts approaches saturation. Then the only way the capacity of the network can be increased is by increasing the capacity of at least one link in the cut or by adding links across the cut. These ways form the basis for the cut-saturation algorithm. The algorithm attempts to satisfy throughput and response time requirements while keeping overall cost low. It has been successfully applied to the various stages of the ARPANET and the AUTODIN II, among others, and has been shown to be near optimum for problems under consideration.[18]

Another class of heuristic algorithms for designing packet switching networks are called concave branch elimination algorithms.[7,18,19] However, their limited applicability makes them less useful than the cut saturation approach.

However, the cut saturation algorithm as given in Reference 7 is somewhat biased by the ARPANET environment and is limited in its applicability to more general packet switching networks. Specifically, (1) it always starts with a good starting design and uses the optimization algorithm to improve upon it, (in a general environment, a "good" starting design is not always available.); (2) it pays little attention to the applicability of multiple capacities and parallel links, (with the proliferation of line tariffs, such considerations are a must.); (3) the criterion to determine the locations to add links across a cut is derived purely from the experience of ARPANET network structure, (therefore, it is likely not well suited to other environments). A unified algorithm for packet switching network optimization has been developed with those deficiencies eliminated.[20]

Step 1—Generating a starting network

(a) Let TRM(k)=kth largest element in {TR(i,j)} where TR(i,j) is the traffic requirement from packet switch i to j.
(b) Reorder first N_1 elements in TRM according to ascending link costs. (N_1 is determined by a prespecified parameter A, such that TRM(1)/TRM(N_1)≤A< TRM(1)/TRM(N_1+1). Let the vector be TRCOST.
(c) Let the end points of the line represented by TRCOST (1) be i and j. If the sum of the line capacity connected to k divided by ΣTR(k,i), $k=i$ or j, does not exceed a preset limit, B, line (i,j) is added. Otherwise, eliminate this link from the list and a new link is picked from TRM and TRCOST is reordered.
(d) Process (c) is repeated until all the nodes are connected.

Parameters A and B impact on computer running time. For small network, they may be set to infinity. For large network, they should be set small. But if too small, a feasible network cannot be realized.

Step 2—Define generalized link capacities

Let ($S_i \mid i=1, \ldots N_S$) be the set of available link speeds (capacities). The set of generalized capacities ($C_m \mid C_m<C_{m+1}$, $m=0,1,2, \ldots$) are defined as:

$$C_0=0$$

$$C_m=\sum_{i=1}^{N_s} m_i S_i \qquad m=1,2,3, \ldots$$

A link can assume a generalized link capacity C_m. C_m, in general, consists of parallel lines of different speeds or capacities. Among them there are m_i parallel lines with speed S_i, for $i=1,2, \ldots, N_s$. The value of m_i can be zero.

Step 3—Routing

The routing operation checks whether a network can support the required throughput or can support more traffic than the original network. It is performed after each network modification and is used to generate a new optimal link flow. The routing operation is a necessary step regardless of which topological optimization technique is used.[21]

Step 4—Saturated cutset determination

The cutset that has been newly saturated must be determined after each operation. Link flows are calculated. The minimal cutset that has the highest utilization is the saturated cutset.

Step 5—Add-only algorithm

If the network structure under consideration cannot accommodate the required throughput, an add-only algorithm determines the "best" link to be added across the two components separated by the saturated cutsets.

Let C_{n_k} be the generalized capacity on link k. The 'adding' of a link is used in a generic sense to mean the upgrading of the value of capacity for link k from C_{n_k} to C_{n_k+1} or higher. If link k did not exist before, this operation becomes the addition of a new link. Otherwise, this operation may upgrade the capacity of individual lines in link k or add parallel lines to link k.

Let $D_k(C_m)$ be the line cost of link k if link k assumes a capacity value of C_m. Link k must be either an existing or a potential new link across the saturated cut. Either of the following two criteria may be used:

Criterion 1. Upgrade the link whose upgrading results in minimum additional costs, that is, upgrade link K if

$$D_K(C_{n_K+1})-D_K(C_{n_K}) = \min_k\{D_k(C_{n_k+1})-D_k(C_{n_k})\}$$

Criterion 2. Upgrade the link whose upgrading appears to result in minimum additional costs per bit, that is, upgrade link K if

$$((D_K(C_{n_K+1})-D_K(C_{n_K}))/(C_{n_K+1}-C_{n_K}) = \min((D_k(C_{n_k+1})-D_k(C_{n_k}))/(C_{n_k+1}-C_n)$$

For computer running time efficiency, only a subset of possible k's should be considered. In particular, choose only those links whose end points have high unsatisfied traffic requirements for nodes across the saturated cut.

Step 6—Delete-only algorithm

This algorithm may be applied if the throughput supported by the network structure under consideration is higher than the requirement. The Delete-Only operation begins with a highly connected topology and "deletes" one "link" at each iteration, continuously reducing cost and throughput. The deleting of link k is used in a general sense to mean the degrading of link k's generalized capacity from C_{n_k} to C_{n_k-1} or lower. If C_{n_k-1} is zero, this operation eliminates link k. Otherwise, this operation reduces the capacities of individual lines in link k or reduces the number of parallel lines in link k.

Either of the following two criteria may be used for "degrading."

Criterion 1. Degrade the link which has the most residue capacity, that is, degrade link K if $C_{n_K-1}-f_K=\max_k(C_{n_k-1}-f_k)>0$ where f_k is traffic flow on link k.

Criterion 2. Degrade the link whose degrading will result in the maximum costs per bit savings, that is, degrade link K if

$$(D_K(C_{n_K})-D_K(C_{n_K-1}))/(f_K-C_{n_K-1}) = \max\{(D_k(C_{n_k})-D_k(C_{n_k-1}))/(f_k-C_{n_k-1})\}$$

for $f_k<C_{n_k-1}$

Step 7—Exchange

This operation combines Add-Only and Delete-Only operations and is used to improve the throughput or the cost. One link is introduced according to the Add-Only algorithm and one link is removed according to the Delete-Only algorithm. Perturbation is performed when the Add-Only algorithm cannot further improve appreciably, the throughput and the Delete-Only algorithm cannot further improve appreciably the cost.

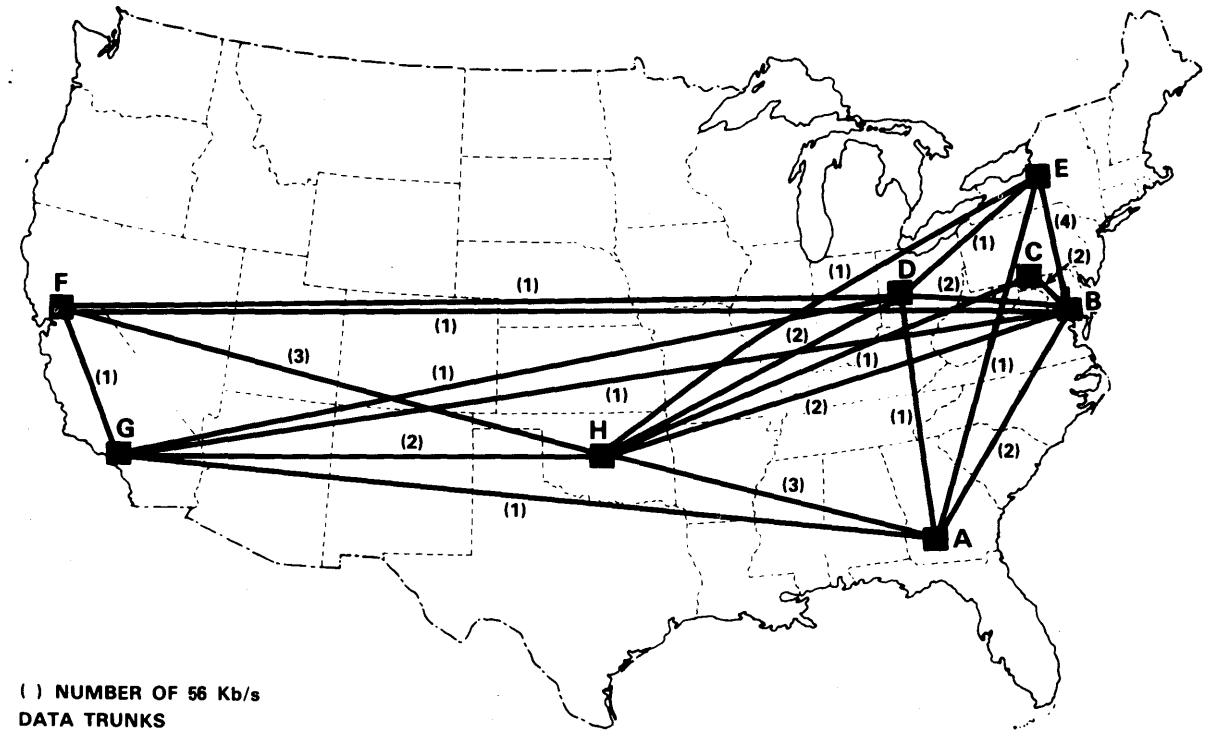
Figure 8 demonstrates the efficiency and effectiveness of the unified algorithm. Figure 8(b) is an AUTODIN II network design that is designed with several executions of ARPANET-based optimization algorithm, and with substantial amount of human interface, including the human configuration of starting network design. Under the same throughput and response time requirements, Figure 8(a) is the design obtained by a single execution of the unified algorithm without input specifying a starting network. It was designed with 28 sec of CPU time on an IBM 370/168 machine which results in a computer charge of $2.20. Yet this design realizes savings of over $200K (11 percent) annually. Figure 8(c) summarizes the comparison.

INTEGRATED OPTIMIZATION AND NAF PLACEMENT

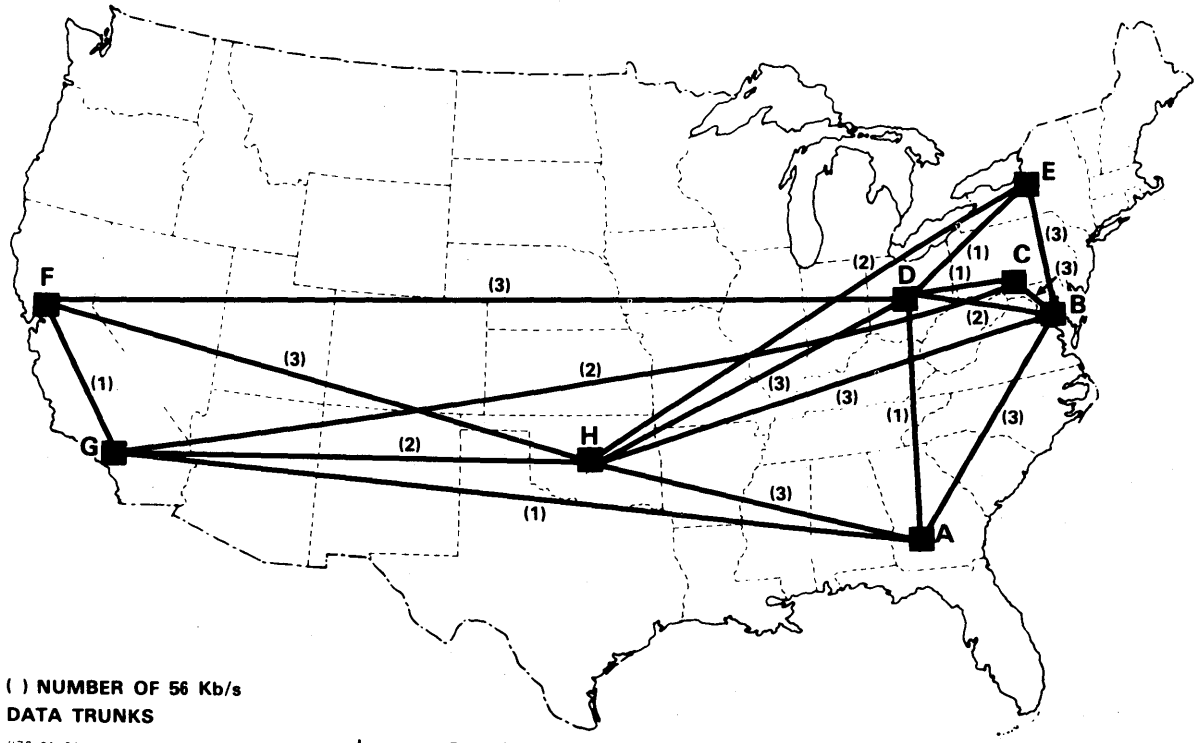
Integrated optimization process

The communications cost for a distributed system in general consists of the costs of NAF's, local access network, and the backbone network. Minimizing the costs for one of the three elements is likely to increase the overall cost. It is important that to optimize the overall communication network costs, the three design problems should be treated as an integrated one. All conventional approaches, however, have treated them as separate sub-optimization or design problems resulting in potentially an overall non-optimized network. We present here an integrated approach.

Since the number and locations impact directly on the costs of both local access and backbone networks, it is only logical to implement the integrated optimization process in conjunction with the optimal placement process of the NAF's. Figure 9 is a simplified version illustrating the interrelation of the integrated process and the three optimization processes. Basically, whenever a set of locations are considered as potential NAF sites, the integrated process interfaces the local access optimization and the backbone



a. Design with Unified Algorithm



b. Design Using ARPANET-based Algorithm and Human Intervention

Figure 8—Comparison of network design with and without the unified algorithm

	Unified Algorithm	ARPANET Algorithm
Network Throughput	1529.2kb/sec	1635.434kb/sec
Avg. End-to-End Delay	0.1233 sec	0.1233 sec.
Comm. Cost/Yr.	$1,675,411	$1,881,474
Computation Time	28 sec (IBM 370/168)	Substantially More
Computer Time Charge	$2.20	Substantially More
Human Interface	None except input preparation	Substantially More

c. Comparison of Designs (a) and (b)

network optimization process to determine the costs of the two subnetworks under this set of NAF's. The network costs may either be determined by a complete optimization procedure or by heuristic estimation. The next section describes the algorithm combining the integrated process and the NAF optimization process.

Optimization of NAF placement

The problem of determining the number and location of NAF's is very similar to the classic warehouse (or plant) location problem. The warehouse location problem is to determine the number, location, and capacity of source sites which minimize the cost of satisfying shipping requirements.[31] A "warehouse" corresponds to an "NAF," and shipping requirements correspond to traffic requirements. Many techniques and algorithms for warehouse location problems have been applied by many authors to the NAF problem.

A general NAF location algorithm has (heuristic) procedures for as many as four problems: determining potential sites, iteratively placing NAF's at the potential sites, estimating the cost of local access networks, and estimating the cost of backbone network.

If the number of sites on which NAF's may be placed is small, every one is designated as a potential site. If not, only a subset of all the possible sites are designated as potential sites to reduce the computational complexity. Iterative procedures are then used to determine the number of NAF's and the eventual NAF sites among the potential NAF sites. The optimal number and locations of NAF's depend on the costs of the network because NAF's interface local access networks and backbone networks. These network costs are sometimes estimated instead of being determined exactly.

Many algorithms have been developed and published, e.g., References 22 through 28. However, they all have only limited applicability. This is probably because there has been very little demand for using algorithms to place the NAF's. There are very few network designs in which the NAF locations have been determined by any of the algorithms referenced above.

For large communication based distributed data processing systems, more flexible and more effective algorithms are needed. In particular, there are the needs for better evaluation of both local access and backbone network costs, and the needs of more flexible algorithms in considering the alternative NAF site. Described below is an algorithm that has incorporated these needs.[29]

The integrated NAF location algorithm

Step 1—Select a subset of permissible sites as potential NAF sites

a. Check the size of locations permissible as NAF sites. Certain locations may be designated as mandatory NAF sites and certain others as permissible. If the sum of the two sets is reasonably small, they are all designated as potential NAF (PNAF) sites and go to Step 2.

b. Cluster Terminals—If the input specified PNAF set is large, the magnitude of the design problem may be reduced by initially clustering the terminals. Intuitively, a good location for a NAF is the center of a cluster of terminals. Thus, by only considering the centers of a set of clusters, the magnitude of the initial problem may be significantly reduced. The number of initial clusters and the cluster size is either input or

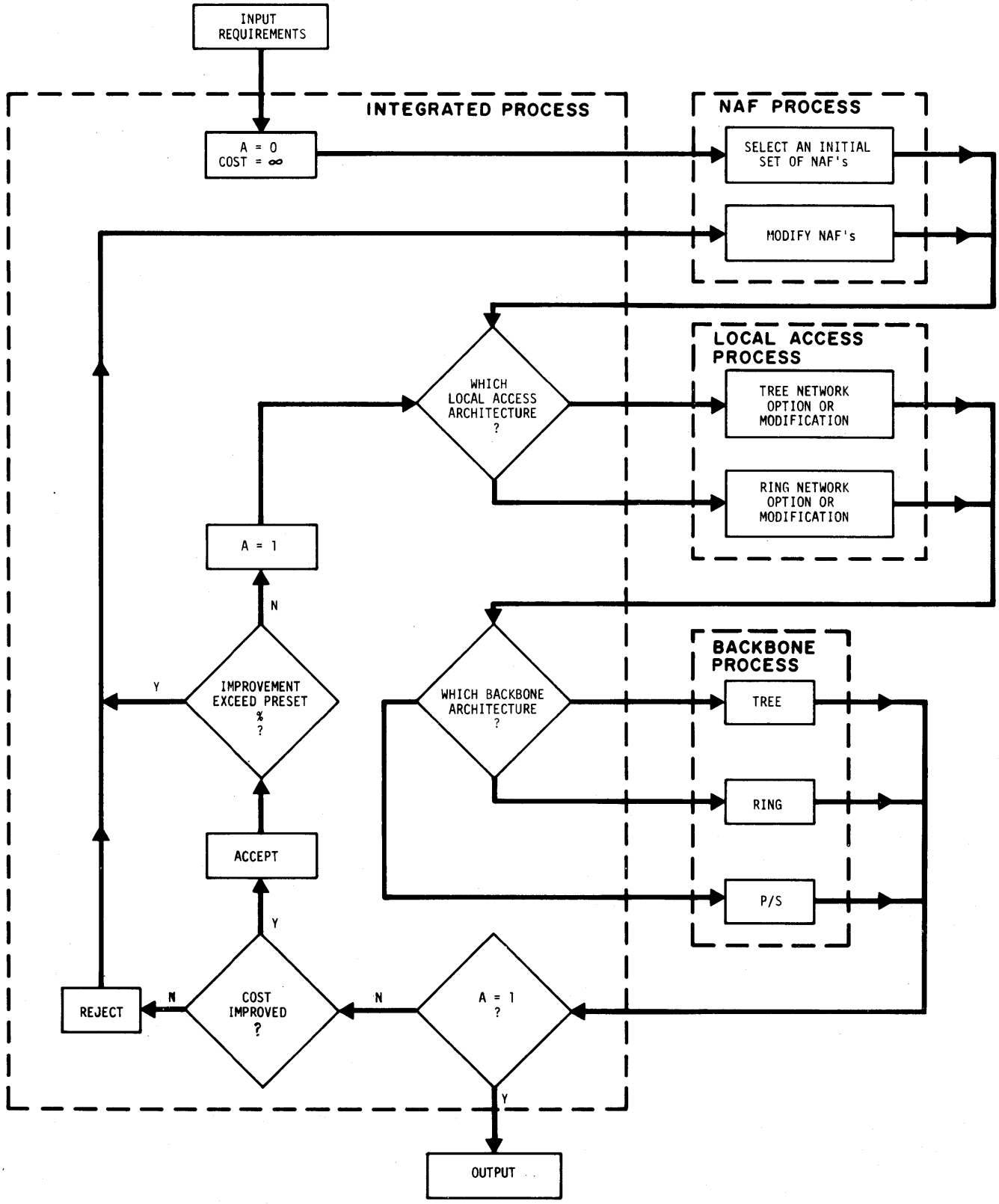


Figure 9—Integrated optimization process

estimated by the program. Then, using Kruskal's algorithm,[8] the clustering is performed.

c. Determine Center of Mass of the Clusters—The center-of-mass (COM's) for the clusters are determined. Only those clusters that have permissible NAF sites but not mandatory NAF sites need be considered. For each such cluster, define the COM by:

$$\text{V Coordinate of COM} = \sum_{1}^{Nc} t_i V_i / \sum t_i$$

$$\text{H Coordinate of COM} = \sum_{1}^{Nc} t_i H_i / \sum t_i$$

where t_i=traffic from Node i

V_i, H_i=(V, H) coordinates of Node i

N_c=Number of nodes in cluster

d. Refine the Center of Mass—The COM as defined above is a mathematically defined location and, hence, may not correspond to any physical node location. Determine a permissible NAF site closest to each COM and assign it as a PNAF.

Step 2—Design or estimate local access network

This step designs the local access network. Initially, assume that all PNAF locations are NAF locations. Using algorithms in earlier sections, the local access network is designed.

Step 3—Design or estimate backbone network

Having designed the local access network, the traffic emanating from each PNAF can be assessed. Four distinct types of backbone network designs are considered: (a) Point-to-point (b) Multipoint (c) Packet-switched (d) Ring.

In (a), for each NAF, the closest CPU is connected.

In (b), the NAF's are connected to the CPU's via multipoint trees using similar algorithms defined earlier.

In (c), Packet-switching techniques are used for the backbone network using algorithms defined earlier.

Step 4—Select the NAF sites

a. Drop Algorithm—In this step, for each PNAF that is non-mandatory and has been selected as a NAF site in the current design step, the potential cost-savings to be obtained by dropping it is evaluated. This savings consists of: (a) savings in backbone network, (b) savings in local access network. A PNAF that yields maximum positive savings is permanently dropped from the network. If the savings is negative, this step is concluded. The savings (a) and (b) can be either evaluated exactly or estimated using heuristics.

- Heuristic Cost Estimate for Backbone
  If the PNAF considered for dropping is singly connected to another node, the backbone cost saving simply equals the line cost plus the physical cost of the NAF. If the PNAF is multiply-connected, consider all the nodes directly connected to this PNAF. Consider, in turn, each of these nodes for direct connection to the rest of the nodes. Whichever of these yields the least cost is considered the alternative. The cost-saving equals the line cost of the new configuration minus the line cost of the previous configuration plus the NAF cost.
- Heuristic Cost Estimate for Local Access
  Consider all the lines that were connected to this PNAF. For each of these lines determine the closest NAF (other than the NAF under consideration) and check if this NAF can accommodate one additional input port. If not, dropping the NAF is infeasible. If it can, consider this the alternative local access configuration. As before, the cost savings equals the new line cost minus the previous line cost.

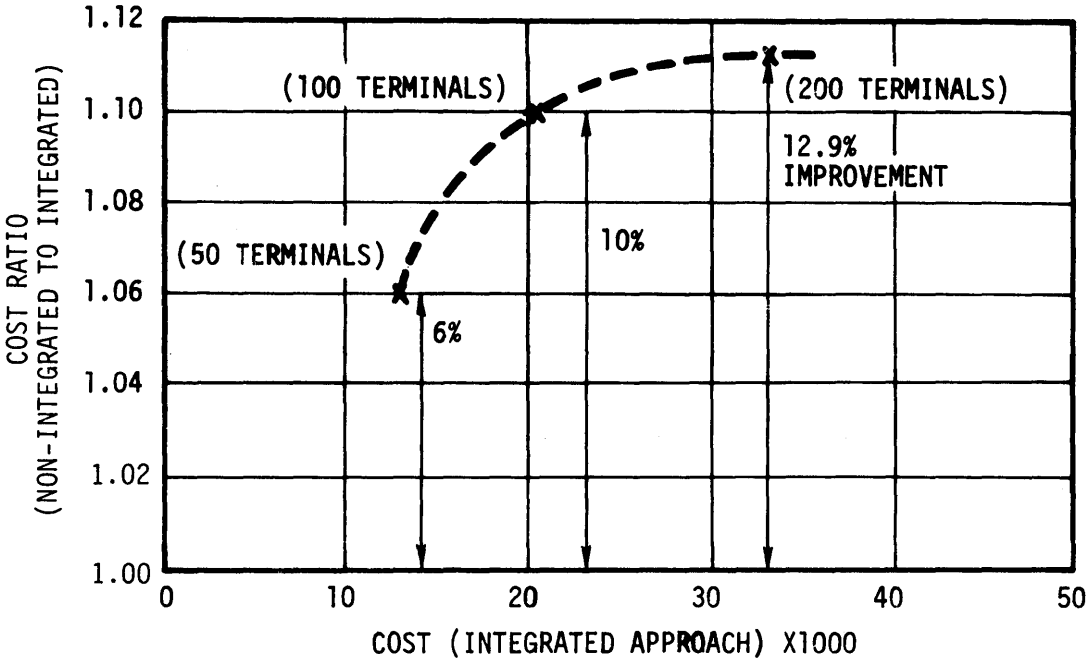
After each K ("K" is an input parameter) iterations, the local access and backbone networks are reoptimized.

b. Add Algorithm—This step considers the cost savings to be obtained by adding an NAF site. For this purpose, PNAF's that are not selected for the current design and that are farthest from the current set of PNAF's are considered for potential addition. The cost savings are heuristically estimated. The PNAF whose addition results in the greatest positive saving is added to the network. If the saving is negative, conclude this step. If the number of iterations equals K, the local access and backbone networks are redesigned.

c. Split Algorithm—This step considers the cost saving that results if a PNAF having a large number of terminals is replaced by two PNAF's. Consider "KS" (input parameter) PNAF's that have the most number of terminals. For each of these, heuristically estimate the cost saving by replacing it with two NAF's. If the maximum cost saving is positive, the split is made permanent; if not, conclude this step. After K iterations, the local access and backbone networks are redesigned.

d. Exchange Algorithm—This step examines the cost saving obtainable by exchanging NAF locations. It considers unselected PNAF locations that are farthest from the current set of selected PNAF's; each of these locations is examined for the effect of exchanging it with a selected PNAF closest to it. As before, an exchange which yields the most positive heuristic cost saving estimate is made permanent. After each K iteration, the local access and backbone networks are redesigned.

e. Merging Algorithm—This step examines the cost saving obtainable by merging a PNAF which has a small number of terminals associated with it, with a neigh-



(a) Cost Comparison Curve

	INAF DESIGN	COM DESIGN	ATD DESIGN
NO. OF SWITCHES	1	1	1
NO. OF TERMINALS	50	50	50
ANNUAL TOTAL COST	$110,248	$117,048	$124,908
NO. OF SWITCHES	1	1	1
NO. OF TERMINALS	100	100	100
ANNUAL TOTAL COST	$159,120	$175,536	$187,344
NO. OF SWITCHES	1	1	1
NO. OF TERMINALS	200	200	200
ANNUAL TOTAL COST	$247,860	$276,876	$277,956

(b) Cost Comparison Table

	COST BASIS
CONCENTRATOR (NAF)	$200/MONTH
LINE CHARGES/MI.	$0.50/MILE
TERMINATION CHARGE	$$40/TERMINATION

(c) Charges Considered in Determining Communication Cost

Figure 10—Cost comparison information

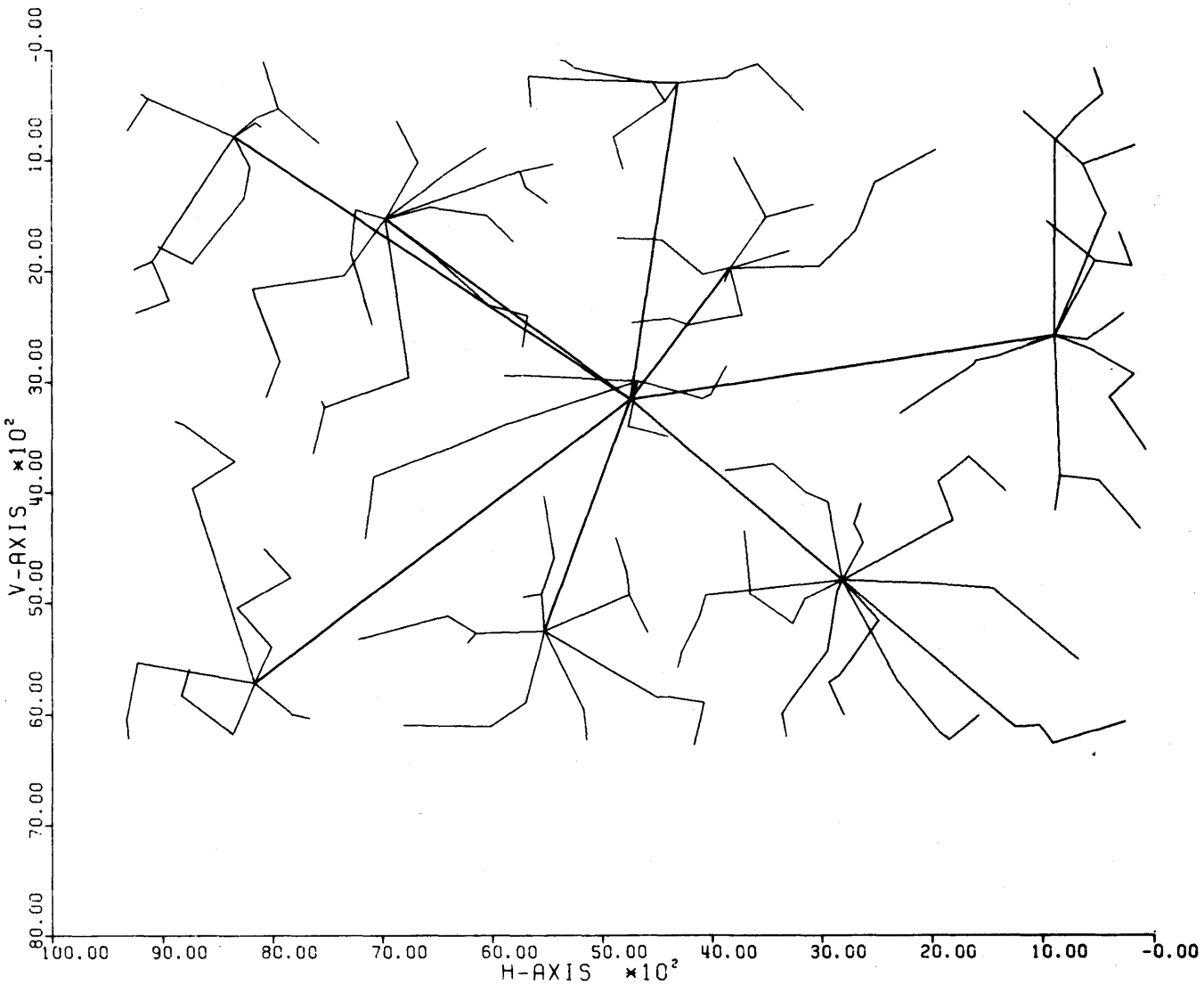


Figure 11—Integrated algorithm network design (200 terminals)

boring PNAF which can accommodate these additional terminals.

In any one design run, all substeps need not be executed. For a given design problem one of these steps may be more useful in achieving low cost networks than others. The designer can experiment with the order and number of these heuristics that need to be called to yield a near-optimum design.

Step 5—Refinement and reoptimization

If any network adjustments have been made since last redesigning the network, reoptimize both the local access and backbone networks.

Figure 10a compares the costs between the designs obtained from the integrated and non-integrated approaches. Cost information for the non-integrated approach was derived from data presented in the recent McGregor-Shen paper.[28] Only local access, backbone line cost, and concentrator costs were considered in doing the comparative evaluation, since these costs are used primarily in our optimization process. Fixed termination charges of $40/termination were subtracted from the McGregor-Shen data in order to provide a basis for comparison.

Locations of terminals used in the data presented in Figure 10 were obtained from a uniformly random distribution of sites taken over a 2000 by 3000 mile rectangle. This distribution of terminals was provided again in order to have a basis for comparison between our integrated algorithm and the algorithm presented in the McGregor-Shen paper.[28]

The specific cost comparisons of the two approaches for three network designs are given in the table in Figure 10b. Figure 10c gives the cost information on lines and devices. Figure 11 is the plot of one of the network designs (200 terminals design). In the above comparisons it should be noted that the reason for using McGregor-Shen's data is that

the authors felt that the McGregor-Shen COM algorithm was demonstrated to be the best heuristic algorithm published to date. Based on our comparative analysis we feel that our integrated algorithm approach improves upon the McGregor-Shen algorithm and uniformly outperforms all other non-integrated approaches published to date.

CONCLUSION

With the fast growth in the communication-based distributed data processing systems, the emphasis on the objectives of the optimization has shifted. We have presented a set of integrated network optimization algorithms that are best suited for this environment. Some of the algorithms are newly developed; some are the modifications of the existing ones.

There are needs for reducing, substantially, the computation time for local access networks with a large number of terminals. We have developed two implementation schemes, heuristic in nature, to satisfy these needs. The computation time, with these schemes, grows practically linearly as the number of terminal locations grows; and, in practical networks with more than two or three hundred terminals, the computation time actually grows less than linearly.

There are some needs for topological optimization algorithms for ring networks, which have not been available in the open literature in the data communication field. We presented such algorithms, developed by modifications of works in the graph theory area.

There are needs for improvements on the widely used packet switching network optimization algorithms that were developed with ARPA support. The improvements are to eliminate its bias from the ARPA environment and to improve the design results and computation times. We presented a generalized algorithm to achieve these improvements.

There are needs for more effective algorithm than currently available on the determination of the number and locations of network access facilities. To realize these needs, we presented an algorithm that adopts many good features of the existing algorithms and is augmented with new ones.

Finally, there are the needs for integrating the different optimization processes into one unified process, which would turn out better design results. We presented such a process in conjunction with the NAF algorithms. In the program that we have developed with the implementation of these algorithms we were also able to integrate into the same process the different network structures (tree, ring, mesh) and variations associated with differences in line tariffs.

ACKNOWLEDGMENT

The authors wish to express their appreciation to J. McLeod and R. A. Pickens for their encouragement, to D. Sapir for informative discussions on Backbone Network design and L. Gerke for her efforts in the development of the computer programs.

REFERENCES

1. Chou, W., "Computer Communications Networks—Parts Make Up the Whole," *Proc. of 1975 NCC.*
2. Esau, L. R. and K. C. Williams, "On Teleprocessing System Design, Part II," *IBM Syst. Journal,* Vol. 5, No. 3, 1966, pp. 142-147.
3. Chou, W. and A. Kershenbaum, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks," *Proc. of the Third Data Communication Symposium,* November, 1973, also *IEEE Trans. on Communications,* November 1974.
4. Farber, D. J., etc., "The Distributed Computing System," Digest of Paper, COMPCON 73.
5. Lin, S., "Computer Solutions of the Traveling Salesman Problem," *BSTJ,* 44, 1965, pp. 2245-2269.
6. Clarke, G., and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *ORSA,* December 1964.
7. Gerla, M., H. Frank, W. Chou, and J. Eckl, "A Cut Saturation Algorithm for Topological Design of Packet-Switched Comm. Networks," *Proc. of 1974 NTC.*
8. Kruskal, J. B., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proc. Amer., Math. Soc.,* Vol. 7, 1956.
9. Prim, R. C., "Shortest Connection Networks and Some Generalizations," *Bell System Tech. J.,* Vol. 36, 1957, pp. 1389-1401.
10. Sharma, R. L. and M. T. El-Bardai, "Suboptimal Communication Network Synthesis," in *Proc. 1970 Int. Conf. Communications,* pp. 19.11-19.16.
11. Karnaugh, M., "A New Class of Algorithms for Multipoint Network Optimization," *Proc. of 1975 ICC.*
12. Chandy, K. M. and R. A. Russell, "The Design of Multipoint Linkages in a Teleprocessing Tree Network," *IEEE Trans. Computers,* Vol. C-21, October 1972, pp. 1062-1066.
13. Cain, J. B., "Centralized Network Design with Multidrop Lines," Memo Report No. 47, Harris Corporation, May 1974.
14. Whitney, V. K. M., "Comparison of Network Topology Optimization Algorithms," in *Proc. 1970 Int. Conf. Communications,* pp. 332-337.
15. Frank, H., I. T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," in *Proc. Spring Joint Computer Conference,* 1970, pp. 581-587.
16. Frank H. and W. Chou, "Topological Optimization of Computer Networks," *Proc. IEEE,* Vol. 60, November 1972, pp. 1385-1397.
17. Lavi, A., and Manning, L. E., "Topological Optimization of Packet-Switched Comm. Networks," CCNG Report E-35, University of Waterloo, October 1975.
18. Gerla, M. and L. Kleinrock, "On the Topological Design of Distributed Computer Networks," *IEEE Trans. on Comm.,* Jan., 1977.
19. Yaged, B. Jr., "Minimum Cost Routing for Static Network Models," *Networks,* Vol. 1, 1971, pp. 139-172.
20. Chou, W. and D. Sapir, "A Unified Algorithm for Packet Switching Network Optimization," in preparation, to be published in 1978.
21. Chou, W. and H. Frank, "Routing Strategies in Computer Networks," Proc. Symp. Comp-Comm. *Networks and Traffic,* Polytechnic Press, N.Y. 1972.
22. Bahl, L. R. and D. T. Tang, "Optimization of Concentrator Locations in Teleprocessing Networks," in *Proc. Symp. Computer-Communication Networks and Teletraffic,* Polytechnic Institute of Brooklyn, Brooklyn, N.Y., April 1972.
23. Woo, L. S. and D. T. Tang, "Optimization of Teleprocessing Networks with Concentrators," *Proc. of 1973 National Telecommunications Conference.*
24. Greenberg, D. A., "A New Approach for the Optimal Placement of Concentrators in a Remote Terminal Communications Network," *National Telecommunications Conference,* November 26-28, 1973, Atlanta, Georgia, pp. 37D-1—37D-7.
25. Dirilter, H. and R. W. Donaldson, "Topological Design of Teleprocessing Networks Using Linear Regression Clustering," *IEEE Trans. on Comm.,* October 1976.

26. Hsieh, W., M. Gerla, P. McGregor, and J. Eckl, "Locating Backbase Switches in a Large Packet Network," *1976 ICCC.*
27. Hsieh, W., "Backbone Switch Location Problem for Mixed Large Packet Switch Networks," *1976 NTC.*
28. McGregor, M. and D. Shen, "Locating Concentration Points in Data Communication Networking," *Proc. of 4th Data Communications Symp.,* November 1975. *IEEE Trans on Comm.,* January 1977.
29. Chou, W., F. Ferrante, M. Balagangadhar, and L. Gerke, "An Integrated Approach to Optimally Locating Network Access Facilities," *Proc. of Fourth International Conference on Computer Communications,* Sept. 1978, Kyoto.
30. Cerf, V. G., D. D. Cowan, R. C. Mullin, and R. G. Stanton, "A Lower Bound on the Average Shortest Path Length in Regular Graphs," *Networks,* Vol. 4, No. 4, 1974.
31. Cooper, L., "Location-Allocation Problems," *Operations Research,* 1962, pp. 331-343.
32. Fratta, L., M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Comm. Network Design," *Networks* Vol. 3, No. 2, 1973.
33. Karnaugh, M., "Multipoint Network Layout Program," International Business Machines Corporation, Rep. RC3723, 1972.

An extensible distributed data base system*

by MAMORU MAEKAWA and SATORU ISHII
Toshiba Research and Development Center
Kawasaki, Japan

INTRODUCTION

Extensibility is one of many claims of improved performance being made for distributed processing systems. Extensibility means system properties, such as "incremental growth and configuration flexibility," "highly adaptable to changes in workload," "incremental replacement and/or upgrading of components (both hardware and software)" and "easily expanded in both capacity and function."[2] Although the need for extensibility is strongly felt, research results for it appear to be still far from satisfactory.[3,4]

A distributed data base system is one example of distributed processing. Although its usefulness is well recognized, the research in this area is also still very much in its infancy. MINIMET, a transaction-oriented homogeneous distributed data base system, has contributed towards the understanding of the distributed data base system.[6,9] The MINIMET research put more emphasis on the lower level of system control so that any data model can be built on it.

The data base system described in this paper is a relational data base, one of whose characteristics, among others, is capacity extensibility.

Extensibility, in this paper, means the following three things:

(a) Physical extensibility
(b) Logical extensibility and data independency
(c) Minimal performance degradation in data base growth.

Physical extensibility means that the system can physically be extended to meet a workload increase. Logical extensibility and data independency mean that the system can logically be extended and that application programs are not affected by system extension. The third characteristic, minimal performance degradation, means that degradation in performance, response times in particular, in the growth of data base is kept minimal. Because of these characteristics, the system will be called an Extensible Distributed Data base System (EDDS).

For physical extensibility, it is apparent that the less hardware for interconnections is provided the more extensible the system is. Obviously, a system consisting of independent subsystems is then most extensible. However, this architecture provides no advantages in coordinating various kinds of information. Distributed systems must consist of autonomous, but not totally independent, cooperating subsystems. The next most extensible architecture would be a common bus system, as shown in Figure 1, in that a number of subsystems are connected by an intersubsystem bus. Any more complex architecture would impose a greater difficulty in expanding the system. Particularly, a shared memory causes a great difficulty.

In the above common bus architecture, the bus capacity is most important in determining system performance. As the workload of the system and the number of subsystems increase, the bus capacity must also be increased. One method to increase bus capacity is to add additional buses. Another is to replace the bus with a faster one. The first method is architecturally more complex than the second, but it has the advantage that no replacements are necessary to increase the capacity. The second method requires bus replacement, but it is architecturally simpler. Both approaches are possible. Further details will not be gone into, it will simply be assumed that a common bus architecture is the basis of the present system.

Another important aspect of extensibility is performance degradation in data base growth. There are two major areas which cause performance degradation; input/output channels and the system itself. In order to avoid performance degradation due to the limitation in input/output channel capacity, it is assumed that the system is connected by n input/output channels to the user systems as shown in Figure 2. This is one of the reasons that a centralized control should be avoided. If there is a centralized control, then any and all data and control information must go through it and the central control may become a major bottleneck.

Like the intersubsystem bus, it is also necessary to try to minimize input/output channel replacement cost. For this purpose, additions rather than replacements are desirable

* This work has been supported by the Agency of Industrial Science and Technology, Ministry of International Trade and Industry, Japan, under the project Pattern Information Processing System.

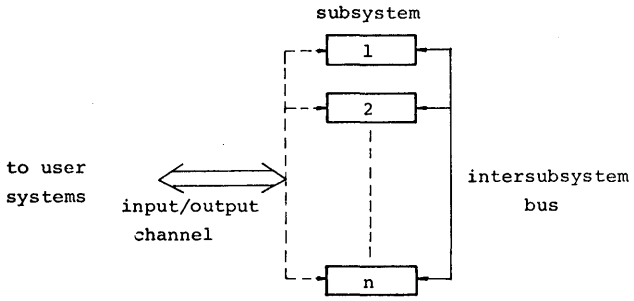


Figure 1—Common bus architecture

when more capacity is needed. Both approaches are feasible and the architecture shown in Figure 2 is assumed to be the basis of discussions.

Performance degradation in the system itself is mainly caused by speed limitation of the intersubsystem bus, which is necessary to transfer data and control information between subsystems to assure proper system operation. The bus capacity is thus an important system parameter. Performance degradation is analyzed in a later section.

As indicated earlier, the system described in this paper is a relational data base. The query language for the system is relational-calculus oriented and has the following three basic statements.

Retrieval
  Get R | condition
  Those tuples that meet the "condition" are retrieved and placed into relation or view "R". The condition is any relational calculus involving relations and views, which are specified only by name.
Relation creation
  Create R | condition
  Relation "R" is created under the "condition".
Relation deletion
  Purge R
  Relation "R" is deleted.

Logical extensibility and data independency are now described in more detail.

LOGICAL EXTENSIBILITY AND DATA INDEPENDENCY

Logical extensibility and data independency mean that neither internal nor external logic is affected by system extension. The approach described in this paper, whereby to obtain this property, is one of the extremes of modular approach. In a general modular approach, a number of modules are prepared and put together to form a particular system. Extensibility is obtained simply by adding more modules. Depending on the size of the modules, the cost of interconnecting them and flexibility of resulting systems vary. The present system requires the following:

(1) The system must be able to work even when it consists of only one subsystem. This implies that the subsystem must be able to perform all the functions required for the data base processing.
(2) No additional programming work, either internal or external, should be required in order to add additional subsystems. This implies that every subsystem must be logically identical.

The EDDS tries to obtain internal logical extensibility in the following way.

(a) Any internal logic is exactly copied and placed in each subsystem. Therefore, no program moves are required. Note that although user programs are allowed to be cataloged and executed at a user request, they are here considered to be data.
(b) Data are separated into two categories: relations and internal data. Relations are made up of information directly created and accessed by users. Internal data are the information necessary for assuring proper system operation. These data are created and maintained by the system. Relations keep spreading over the subsystems and only one copy of each is maintained at any time. A single relation can be spread over several subsystems. On the other hand internal data are duplicated and placed in each subsystem. Some examples of internal data are relation sizes, locations and structural characteristics. This information is distributed among subsystems under a loose control. Here, loose control means that the control of mutual exclusions is neglected and, thus, there might be some discrepancy at some time between multiple copies of internal data distributed among subsystems. The correctness of any decisions based on this information is, as a result, probabilistic.

The above discussions focus on the internal logic of the data base system. The external logic's logical extensibility is also very important, which is commonly known as "data independency." Particularly, we are here concerned with the number of subsystems. Data independency requires that accesses to data base be made independently of the internal structure, including the number of subsystems. In order to obtain this data independency, the following basic approach is used.

(a) Each request from a user system is broadcast to every

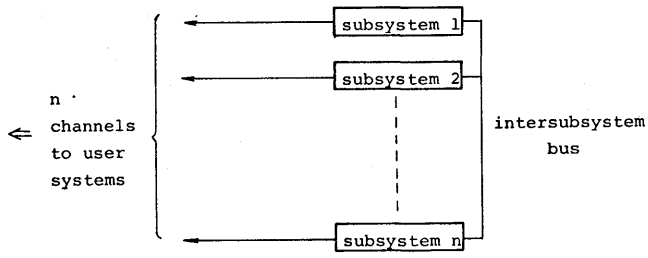


Figure 2—System architecture

subsystem. Note that, at this point, it is not yet known where the necessary relations reside and which subsystem is going to take care of the request. The request itself does not contain any such information. The request is written in the query language mentioned before.

(b) Let every subsystem start to try the processing of the request, but make all the subsystems except the optimal one drop out during the course of processing. Eventually, the request is taken care of by the optimal subsystem. The results are returned to the user system directly from it.

A key characteristic of this approach is expressed by the term "eventually" in the above sentence. No central control exists and no one tells any subsystem when to quit. Each subsystem is autonomous. This approach can be realized by the following algorithm.

Algorithm

1. A user system issues a request. The request is accompanied with the time when it was created. This created time, together with the user system identification, will be used as a unique identification of the request.
2. The request is broadcast to every subsystem in the data base system.
3. The request is placed in the waiting queue of each subsystem.
4. When the turn of the request comes, its subsystem tests whether the processing of the request has already been started by another subsystem. If so, the subsystem simply discards the request; otherwise it analyzes the request and, using some criterion, it determines whether it should accept the request. If it has decided not to do so, then the request is simply discarded. Otherwise, the processing is started. The above criterion will be discussed later.
5. A subsystem now starts processing the request and sends a signal to other subsystems to notify it. If the request involves a creation of a relation, an OPEN to a dummy relation is made prior to any operations necessary for the request.
6. It is assumed that any relation must be OPENed before any accesses are allowed. When an access to a relation is made with an OPEN procedure, a check is made to determine whether an access to it has been made for the same request from another subsystem. If so, the subsystem terminates the processing of this request and simply discards it. Otherwise, it continues the processing. This checking mechanism will also be discussed later.
7. When the results are produced, they are simply returned directly from the subsystem which accepted the request to the user system that issued the request and no other subsystems and/or user systems will be involved.

Next, the acceptance criterion mentioned in (4), notifying mechanism in (5), checking mechanism in (6) and an OPEN to a dummy relation in (4) are discussed.

Acceptance criterion

Each subsystem determines whether it accepts a request or not according to some acceptance criterion.

This acceptance criterion should be aimed at obtaining a balanced load assignment with a minimal overhead. It must assure that at least one subsystem accepts a request. A simplest algorithm is to let each subsystem accept every request and attempt to try its processing. This algorithm is simple and uses no information about the request. We call this algorithm "all selection algorithm" for later references and assume that the queue discipline is first-come, first-served.

The algorithm can be improved by utilizing information about the request. The following information is useful:

(a) Which relations are to be referenced.
(b) How often they are to be referenced.
(c) Which of them are in the subsystem concerned.

Suppose that request R_i references relations $D_{i1}, D_{i2}, \ldots, D_{im}$ with frequency $F_{i1}, F_{i2}, \ldots, F_{im}$, each. Then, the total number of references is

$$\sum_{j=1}^{m} F_{ij}.$$

Assume that a fraction W_{ij} of relation D_{ij} resides in the subsystem concerned. Then, the probability that an access falls into the subsystem concerned is

$$W = \frac{\sum_{j=1}^{m} W_{ij} F_{ij}}{\sum_{j=1}^{m} F_{ij}}$$

A natural acceptance criterion is then:

$$\text{accept if} \quad W \geqq \alpha$$

for some threshold value α. If we take $\alpha=0$, it becomes the all selection algorithm. If we take $\alpha=1$, many requests would be missed and there would be a very high probability that a request would not be served by any subsystem at all. The maximum value that insures the acceptance of a request by at least one subsystem is

$$\frac{1}{n}$$

where n is the number of subsystems. We call the above improved criterion "sufficient selection algorithm" for later references.

Notifying mechanism

The request acceptance notification by a subsystem is not essential for proper system operation, but is employed to reduce overhead. The mechanism is a very simple one. Each

request is labeled with a unique identification, as mentioned before. Whenever a subsystem starts processing, it broadcasts a signal to notify every other subsystem of this. No control is placed on the work of this mechanism. Assurance of the proper operation of this mechanism requires a system-wide or intersubsystem mutual exclusion of signal setting, resetting and checking. Without a shared memory, this implementation of intersubsystem mutual exclusions is a rather time consuming process. Rather than trying to make the mechanism complete, instead it is made probabilistic. In other words, no intersubsystem mutual exclusions are employed, although intrasubsystem mutual exclusions are guaranteed. This simplification causes cases where, due to the delay in signal transferring, a subsystem checks the signal before it arrives. Therefore, a subsystem may proceed in spite of the fact that another subsystem is already processing. This causes some loss in processing power due to duplicated work, but the mechanism itself does not need a difficult control and the architecture can remain simple. Note that a logic can be built so that there are no opposite cases; namely the case that the signal is set while no subsystems actually have started. This guarantees a necessary request acceptance condition wherein a request is always accepted by at least one subsystem.

Checking algorithm

The checking algorithm is the last gate that a request must go through. If a request is passed through this check, then it will be processed through the completion. The checking algorithm is implemented in the OPEN procedure for a relation. Each OPEN procedure has a list of processed requests augmented with their subsystem's names. Whenever an OPEN is requested, it checks whether or not another OPEN for the same request from a different subsystem has already been made. If so, the OPEN procedure returns control indicating this and the subsystem then simply discards the whole request. It is assumed that any relation is always OPENed by its owner subsystem. This assures the mutual exclusion of OPEN procedure and makes the checking mechanism much easier. Note that it is assumed a relation is owned by a single subsystem although it may be spread over several subsystems.

OPEN to a dummy relation

An OPEN to a dummy relation is issued to assure proper relation creation. For relation creations, the following problems must be considered:

(a) How to control a relation creation so that one and only one is properly created.
(b) How to distribute relations in a balanced way.

The first problem is specific to the EDDS, in which a number of subsystems essentially try to do the same operation in regard to the same relation, including a relation creation, at the same time. Thus, when several subsystems try to create a relation, operations must be controlled so that one and only one subsystem will do the job. For this control, an access is defined and attached to a dummy relation to each request that involves a relation creation. It is assumed that this access to the dummy relation is made at the top of processing the request. In this way, through using the above checking algorithm, it will be assured that one and only one subsystem completes the operation and that no extra operations are made. A small problem is how to determine the name of the dummy relation. The naming mechanism should have the characteristics:

(a) Dummy relations are distributed to each subsystem so that the system can operate irrespective of the number of subsystems.
(b) Accesses to the dummy relations are evenly distributed among subsystems.

In order to accomplish these characteristics, the following method is adopted.

(a) A dummy relation is assigned to each subsystem, whose names are subsystem numbers 1 through n.
(b) Each request is accompanied with its created time, t. The name of the dummy relation for this request is computed as (modulo n of t)+1.

In this way, it should be obvious that the required characteristics are achieved, although the total number of subsystems, n, must be known to each subsystem.

The second problem, how to distribute relations in a balanced way, is another important problem. A simple and perhaps natural way to obtain this is to let the first available subsystem accept the request and create the required relation. This method sets up a relation to be assigned to a most likely lightest-loaded subsystem. Another simple and natural way is to assign the relation to the subsystem that has the largest amount of empty space. Control of this method is also easy. However, the former load-oriented approach is taken, since it is easier when it is combined with data search operations.

CORRECTNESS AND OPTIMALITY

This section proves the correctness of the algorithm presented in the previous section and shows that the algorithm is statistically optimal for system balancing or load sharing.

Correctness

Algorithm correctness will be proved by showing:

(a) At least one subsystem accepts a request,
(b) No more than one subsystem accepts a request,
(c) No deadlocks occur
(d) System is alive

Methodology

The methodology used here is a slight extension of Bochmann and Gecsei's method.[1] It is extended by adding remote read and write operations, denoted by READ (a:=v) and WRITE (v:=a). A READ (a:=v) specifies an access to a remote variable v and assigns the value of v to local variable a. Some arbitrary time delay exists between the initiation of the action and an actual reference to variable v. Control, however, stays at the READ command until the action is completed. A WRITE (v:=a) causes a remote variable v to be changed to the value of local variable a. A WRITE action does not wait for the completion of the action and some arbitrary delay exists between initiation and an actual change in v.

Model

For simplicity, it is assumed that the acceptance criterion is the all selection policy and limits the number of subsystems to two. Also, the number of relations used by a request is limited to two. As will be seen, an extension to a general case is very straight forward. A description of each subsystem can be given, referring to the algorithm described in the previous section.

(a) State transition diagram for subsystem k. (see Figure 3)

(b) Variables

Variable

$C_i[1:n]$: integer for each relation i

$$C_i[j]=\begin{cases}\text{Meaning}\\ h & \text{if the } i\text{th relation for the } j\text{th request is OPENed by the } h\text{th subsystem.}\\ \text{null} & \text{otherwise}\end{cases}$$

$S_k[1:n]$: (0,1) for each subsystem k

$$S_k[j]=\begin{cases}1 & \text{if the } j\text{th request is already being processed by some subsystem}\\ 0 & \text{otherwise}\end{cases}$$

(c) Actions

Transition	Enabling predicate	Action	Meaning
Arrival	at least one request is waiting	;	take the next request, say jth, from the waiting queue
Notify	$S_k[j]=0$	$S_k[j]:=1$; WRITE $(S_i[j]:=1)$ for $i=1,2,\ldots,n,\ i\neq k$;	notify the start of processing
Processed	$S_k[j]=1$	;	discard the request
OPEN OK	$C_h[j]=$null $\wedge C_h[m]=$null for any $m\neq j$	$C_h[j]:=k$;	the first OPEN of the jth request is for the hth relation
OPEN Unsuccessful	$C_h[j]\neq$null	;	another subsystem has already OPENed relation h
OPEN Good	$C_i[j]=$null $\wedge C_i[m]=$null for any $m\neq j$	$C_i[j]:=k$;	the first OPEN for the ith relation
OPEN Failure	$C_i[j]\neq$null	;	another subsystem has already OPENed relation i
Return	true	$C_h[j]:=$null; $C_i[j]:=$null;	after the completion restart a new cycle

Verification

The enabling predicates for OPEN OK and OPEN Good involve more than a single request. Because of this, deadlocks can occur between conflicting requests. Since these deadlocks can be avoided by a usual technique, it is assumed they will not be a problem. Then, it is only necessary to analyze the case for the single same request.

Let us denote by (t_1,t_2) the global system state, where t_1 and t_2 are two subsystem states. Possible transitions of the global system can be drawn as shown in Figure 4. Note that, since the subsystems are logically identical, the order of executions of OPEN's for the relations for the same request is the same in each subsystem. From Figure 4, we can prove the four properties of the system.

(a) At least one subsystem accepts a request.
This is verified because once the system makes a transition "Arrival," it always comes to a state involving a subsystem state (5).

(b) No more than one subsystem accepts a request.
This is verified because no (5,5) states are allowed and because once a state $(5,x)$ is reached, then state

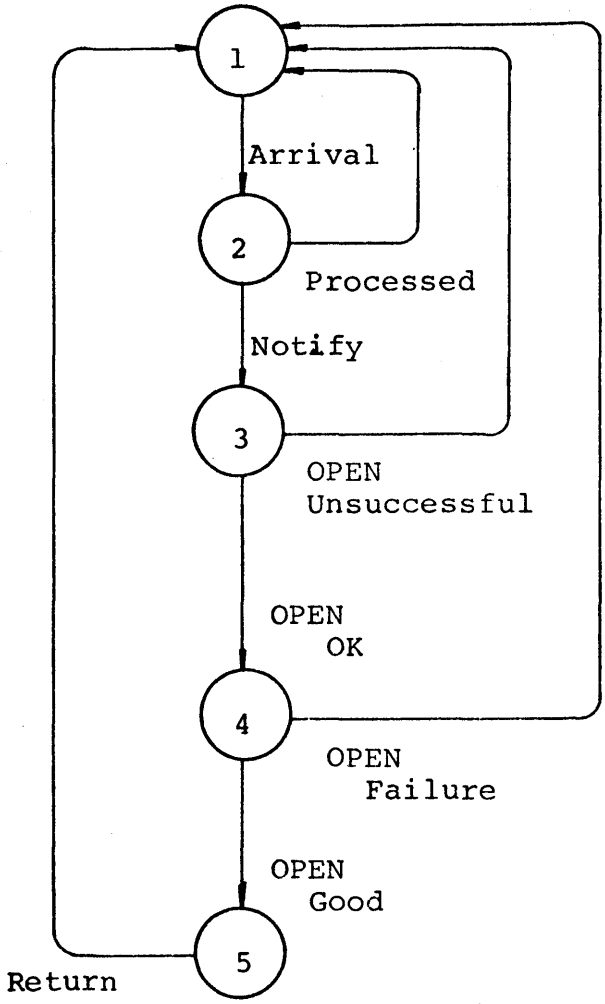


Figure 3—State transition diagram

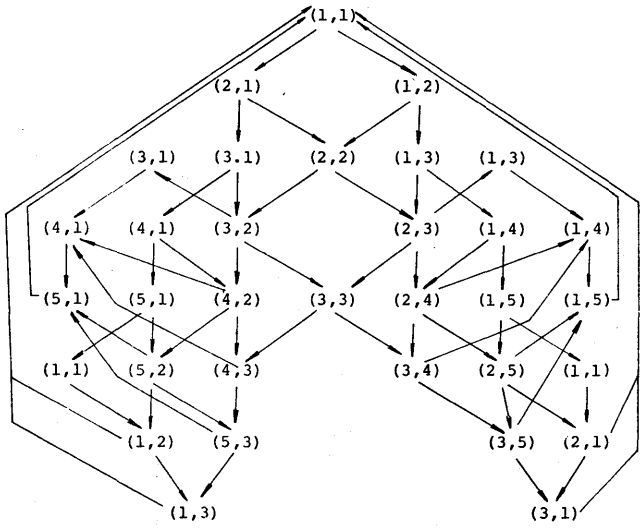


Figure 4—Possible transitions

(y,5) can never be reached for the same request, and vice versa.

(c) No deadlocks occur
Since no states are blocked, no deadlocks occur.

(d) System is alive
This can be shown because, after the system reaches state (5,x) or (y,5) it always returns to the original state (1,1).

The above verification is limited to a special case, but an extension to a general case should be obvious and straightforward.

Optimality

It can be seen that the algorithm described in the previous section contains so called automatic load sharing. Several methods for automatic load sharing have been proposed. LeLann's two methods,[7] for instance, are:

(a) Diffusion technique
(b) Circulating vector technique.

The above two methods and many others exchange information about the load of each subsystem. Based on this information, each subsystem determines whether it accepts a new request. The present method, under the all selection discipline, does not need any information exchange and can be proved to be optimal for load sharing in a statistical sense. The present scheme is free from any consistency problems caused by communication delays, because no information exchanges are required. Note that any scheme that exchanges information may have consistency problems, due to communication delays.

Model and proof

The present method, under the all selection discipline, is statistically equivalent to a single queue with multiple servers, because each request is placed in every queue. Any scheme that exchanges information corresponds to a system of separate-queues with some jockeying. Whatever jockeying is used, they cannot be better than a single queue as far as load sharing is concerned. This can be proved very easily. Some simple analysis for jockeying for a variety of strategies is made by Koenigsburg[5] and a detailed analysis of two queues is made by Maekawa.[8]

PERFORMANCE ANALYSIS

For performance analysis, the data base is simply considered to be a collection of pairs (attribute, value). They are put together to form records, domains and relations. In this section, the only concern is with their statistical characteristics. There is no concern over whether they are actually relations, domains or records.

The present data base system consists of a number of subsystems, as shown in Figure 2. Each subsystem has a

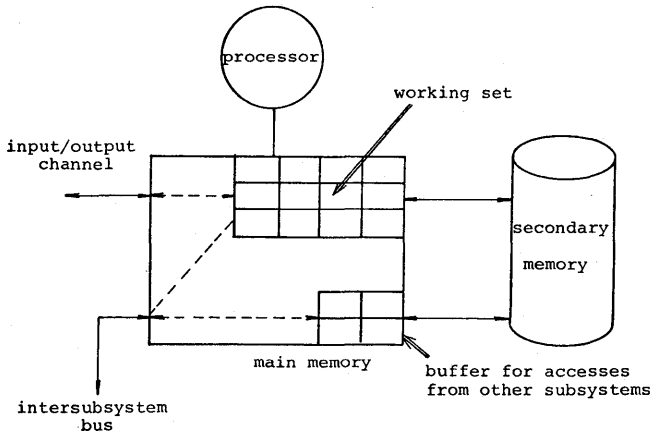


Figure 5—Subsystem

processor, main memory and secondary memory, as shown in Figure 5. In the main memory, there are two kinds of buffers; one is for data handled directly by its own processor and the other for accesses from other subsystems. The former buffer contains a working set of data. It is assumed that both main memory and secondary memory are divided into pages of equal size and that a paging scheme is employed. A model of the whole system is shown in Figure 6.

(1) There are m users and n subsystems connected by n input/output channels
(2) Each user issues a request with rate λ.
(3) A request is first examined by its subsystem. It is discarded unless the subsystem decides to accept it. It is assumed that the initial examination time is governed by random variable A and the probability of acceptance is denoted by a. Probability a is dependent on the system state.

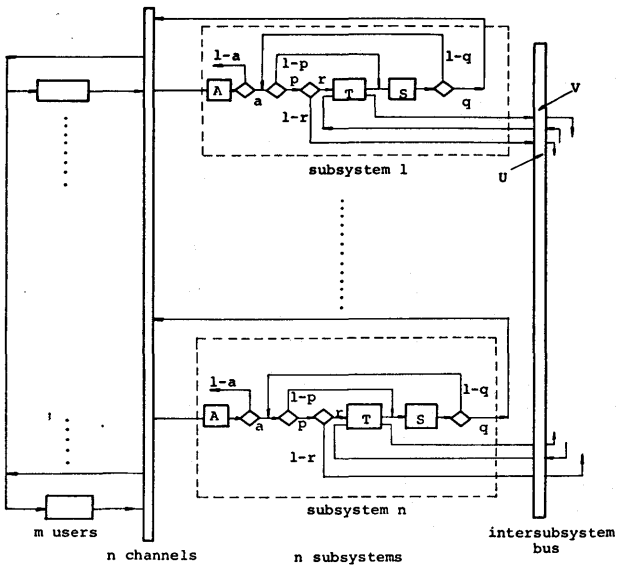


Figure 6—Mathematical model

(4) The processing of a request is a cycle of an access to data, usually a tuple of a relation, and its processing. When data are not in the main memory, they are brought into the main memory from a secondary memory. Time to access data in the secondary memory is assumed governed by random variable T. When data are obtained in the main memory, processing the data is started. Its service time is assumed governed by random variable S. On the completion of the service, request processing completes with probability q and repeats with probability $(1-q)$. Thus, the number of cycles per request is geometrically distributed with mean $1/q$.
(5) The missing page rate is denoted by p. When a page is not already in a main memory, the required page is brought in from its own subsystem's secondary memory or from another subsystem. It is assumed that the probability that a page is brought in from another subsystem is $(1-r)$. If a page is in its own subsystem's secondary memory, it takes only time T to bring the page in. If it is not, however, a request for data acquisition and the obtained data must go through the intersubsystem bus, whose times are governed by random variables U and V, respectively.

The major performance problems in this system are

(a) How much performance degradation is imposed in each subsystem due to this architecture.
(b) How good is load balancing.
(c) How much overhead due to this architecture is imposed.

The performance degradation is, to some extent, controlled by the acceptance criterion. The performance degradation of the system, excluding I/O channels, is mainly due to the following overhead.

(a) An unsuccessful attempt for access to a relation per subsystem for each request may be necessary.
(b) When a page is in another subsystem's secondary memory, it must be brought in through the intersubsystem bus.

Overhead (a) is negligibly small, compared to the total processing necessary for a request. Thus, overhead (b) is the major cause for performance degradation. Instead of analyzing the whole system shown in Figure 6, the performance of each subsystem may be approximated by the model shown in Figure 7. The rationale behind this approximation is obvious. The number of subsystems affects probability r

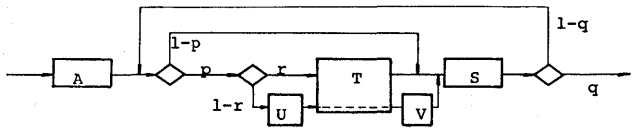


Figure 7—Approximation model

as well as transfer times U and V. As a simple comparison, make a mean-value analysis.

(A) Single-subsystem case
In this case, $r=1$. Then, the mean service time for a request is

$$E[\text{service time}]_1 = E[\text{cycle time}]E[\text{number of cycles}]+E[A] = (pE[T]+E[S])/q+E[A] \quad (2)$$

(B) Multi-subsystem case

$$E[\text{service time}]_m = E[\text{cycle time}]E[\text{number of cycles}]+E[A] = \{pE[T]+p(1-r)(E[U]+E[V]) + E[S]\}/q+E[A] \quad (3)$$

Since $E[A]$ is small, compared to the total processing time, ignore $E[A]$ and then take the ratio of the two for comparison purposes.

$$d=\frac{E[\text{service time}]_m}{E[\text{service time}]_1} = \frac{p+E[S]/E[T]}{p+p(1-r)(E[U]+E[U]+E[V])/E[T]+E[S]/E[T]}$$

Table I shows the values of d for various values of

$$h=(1-r)(E[U]+E[V])/E[T]. \quad (4)$$

It is assumed that $E[U]$, is one-tenth of $E[V]$, page size is 2 kilo-byte, $E[T]$ is 30 ms and (1−r) is 0.3. Then h and c are related by

$$h=(1-r)(E[U]+E[V])/E[T] = 0.022/c.$$

Under normal operation conditions, the missing page rate, p, should be less than 1/100. If it is assumed that $h=0.1$, then performance degradation is 2 to 6 percent. For $h=0.1$, 0.22 mega-byte/second per subsystem is needed. Thus, if a 1 mega-byte/sec bus is provided, then a system, made up of 4 subsystems, should be able to work with 2 to 6 percent performance degradation. When there is a fairly fast secondary memory and $E[S]/E[T]=1/20$ is justified, then, for $h=0.7$, or $c=1$ mega-byte/sec, performance degradation is only 10 percent with 30 subsystems. At any rate, it is observed that the smaller the missing page rate, and the faster the secondary memory, the less performance degradation is incurred and the larger a system can be constructed. Thus, a faster buffer memory between main and secondary memories would greatly improve system performance. A magnetic bubble or CCD memory would be a great help here.

Next, the acceptance criterion effect is analyzed. The two policies discussed before are compared.

(a) All selection policy
(b) Sufficient selection policy

Again, only a mean-value analysis is made. The assumptions are as follows:

(1) The subsystem reference frequency is described by a truncated geometric distribution with parameter $(1-w)$. Namely, a request R_i refers a subsystem S_{ij} with probability

$$f(j)=\frac{w(1-w)^{j-1}}{1-(1-w)^j}, \qquad 0<w\leqq 1. \quad (5)$$

(2) Each subsystem is identified with the number k, $1\leqq k\leqq n$. The kth subsystem is denoted by S_k. The correspondence of S_{ij} to S_k is assumed to be random. Namely, S_k corresponds to S_{ij} with equal probability $1/n$.

(3) The request R_i's processing time by subsystem S_{ij} is denoted by

$$P_{w,j}.$$

Since the system is symmetric and, if a page transfer time between subsystems is assumed independent of

TABLE I.—Performance Degradation

h	c (mega-byte/sec)	$p=1/3$ $E[S]/E[T]=1/20$	$p=1/3$ $E[S]/E[T]=1/200$	$p=1/10$ 1/20	$p=1/10$ 1/200	p=1/100 1/20	p=1/100 1/200	p=1/1000 1/20	p=1/1000 1/200
0.002	11	0.998	0.998	0.999	0.998	1.000	0.999	1.000	1.000
0.02	1.1	0.982	0.981	0.987	0.981	1.000	0.987	1.000	0.997
0.04	0.55	0.966	0.962	0.974	0.963	0.993	0.974	0.999	0.993
0.06	0.37	0.950	0.944	0.962	0.946	0.990	0.962	0.999	0.990
0.08	0.28	0.935	0.927	0.949	0.929	0.987	0.949	0.998	0.987
0.1	0.22	0.920	0.910	0.938	0.913	0.984	0.038	0.998	0.984
0.15	0.15	0.885	0.871	0.909	0.875	0.976	0.909	0.997	0.976
0.2	0.11	0.852	0.835	0.882	0.840	0.968	0.882	0.996	0.968
0.3	0.073	0.793	0.772	0.833	0.778	0.952	0.833	0.994	0.952
0.5	0.044	0.697	0.670	0.750	0.677	0.923	0.750	0.990	0.923
0.7	0.031	0.622	0.592	0.682	0.600	0.900	0.682	0.986	0.896
1.0	0.022	0.535	0.504	0.600	0.512	0.857	0.600	0.981	0.857
1.5	0.015	0.434	0.404	0.500	0.412	0.800	0.500	0.971	0.800
2.0	0.011	0.365	0.337	0.429	0.344	0.750	0.429	0.962	0.750
3.0	0.0073	0.277	0.253	0.333	0.259	0.667	0.333	0.944	0.667
5.0	0.0044	0.187	0.169	0.231	0.174	0.545	0.231	0.911	0.545

TABLE II.—Overhead

w	j=1	j=2	j=4	j=6
0.9	0.1	0.505	0.750	0.833
0.7	0.3	0.545	0.752	0.833

the subsystems chosen, Eq. (3) can be used as $P_{w,j}$ where r is to be replaced with $f(j)$ of Eq. (3).

$$P_{w,j}=\{pE[T]+p(1-f(j))(E[U]+E[V])+E[S]\}/q+E[A]$$

The following observations are made.

(1) Under the all selection policy, each request is equally selected by a subsystem. Thus, the average service time would be

$$\frac{\sum_{j=1}^{n} P_{w,j}}{n}.$$

For a system made up of a single subsystem, service time is $P_{1,j}$, which is described by Eq. (2). The system load would be very well balanced, because each request is assigned to the first subsystem that becomes empty. The assignment is a reversed FIFO.

(2) Under the sufficient selection policy, only k subsystems become candidates for selections, where $1 \leqq j \leqq k$ and k is determined by the policy. Variable k itself is a random variable. The average service time would be

$$\frac{\sum_{j=1}^{k} P_{w,j}}{k}.$$

Under this policy, the load would also be well balanced but there may be some processing time loss, because there are some chances that empty subsystems cannot take unprocessed requests because they are assigned to some other busy subsystems. However this loss is ignored for simplicity and the necessary intersubsystem bus capacity is calculated for k=1, 2, 4 and 6, whose results are shown in Table III. It is assumed that the system consists of six subsystems. In Table II, the ratio

$$e=\frac{\sum_{j=1}^{k}(1-f(j))}{k}$$

is shown. The above ratio e is related to variable h defined by Eq. (4) as

$$h=e(E[U]+E[V])/E[T].$$

From Table III it can be observed that, for p=0.01 and $E[S]/E[T]$=1/200, the system can work with 10 percent degradation with h=0.15 or c=0.247 for j=2 and w=0.9. Thus with an intersubsystem bus whose capacity is 1 mega-byte/sec, the system of four can be constructed. With a bus of 4 mega-byte/sec, a system of 16 can be built. If $E[S]/E[T]$=1/20 can be assumed, h=0.900 or C=0.0529 is sufficient. This allows 19 subsystems for a 2 mega-byte/sec bus.

It should be noted that the above analysis is very simple. A more elaborate and precise analysis is clearly necessary and is now under way. Its results will be reported elsewhere. Relating to this, there are better subsystem selection policies. One of them is, for instance, to use the following criterion:

(a) Compute w, using Eq. (1)
(b) If w is fairly high for a subsystem, say w>0.7, then any subsystem whose value of w is not greater than 0.7 drops out.
(c) The smaller the value of w is, the more subsystems can be assigned.
(d) When w=1/n, assign all the subsystems. Analysis of this policy would require a more elaborate analysis and is not included in this paper.

TABLE III.—Intersubsystem Bus Capacity

	w=0.9				w=0.3			
h	j=1	j=2	j=4	j=6	j=1	j=2	j=4	j=6
0.002	3.665	18.5	27.5	30.6	11.0	20.0	27.6	30.6
0.02	0.367	1.85	2.75	3.06	1.10	2.00	2.76	3.06
0.04	0.183	0.925	1.38	1.53	0.55	1.00	1.38	1.53
0.06	0.122	0.617	0.917	1.02	0.367	0.667	0.918	1.01
0.08	0.0916	0.463	0.688	0.764	0.275	0.500	0.689	0.764
0.1	0.0733	0.370	0.550	0.611	0.220	0.400	0.551	0.611
0.15	0.0489	0.247	0.367	0.407	0.147	0.267	0.367	0.407
0.2	0.0367	0.185	0.275	0.306	0.110	0.200	0.276	0.306
0.3	0.0244	0.123	0.183	0.204	0.0733	0.133	0.184	0.204
0.5	0.0147	0.0740	0.110	0.122	0.0440	0.0800	0.110	0.122
0.7	0.0105	0.0529	0.0786	0.0873	0.0314	0.0571	0.0787	0.0873
1.0	0.00733	0.0370	0.0550	0.0611	0.0220	0.0400	0.0551	0.0611
1.5	0.00489	0.0427	0.0367	0.0407	0.0147	0.0267	0.0367	0.0407
2.0	0.00367	0.0185	0.0275	0.0306	0.0110	0.0200	0.0276	0.0306
3.0	0.00244	0.0123	0.0183	0.0204	0.00733	0.0133	0.0184	0.0204
5.0	0.00147	0.00740	0.0110	0.0122	0.00440	0.00800	0.0110	0.0122

CONCLUDING REMARKS

Extensibility is one of the important system properties of future systems. Among many extensibility aspects, the present work is concerned with extensibility in capacity. Extensibility, in this sense, is first defined and then a scheme to obtain it is proposed and described. The scheme is shown as an algorithm and then its correctness is verified. Extensibility contains automatic load sharing. Scheme optimality is also shown. Performance analysis shows that this scheme can be used to build a fairly large size extensible data base system without much performance degradation.

Extensibility in capacity means three things; physical extensibility, logical extensibility and minimal performance degradation. The key property of the scheme for logical extensibility is that each request is broadcast to each subsystem and is then tried by every subsystem. Eventually, however, one and only one subsystem completes the request. Each subsystem is autonomous and it itself determines when to drop out. In doing this, no exchanges of information about the load of each system are required. For this reason, no consistency problems occur. No system wide mutual exclusions are necessary in this respect, which makes the logic considerably simpler.

The system performance is analyzed using a simple mean-value analysis. It is observed, from this analysis, that a fairly large size data base system can be built, based on the proposed scheme. Necessary capacity of the intersubsystem bus can remain realistic. Any means of reducing missing page ratio would greatly help improve the performance of the system. Here, magnetic bubble or CCD memory can be a great help.

ACKNOWLEDGMENTS

The authors would like to acknowledge Prof. Eric Manning of the University of Waterloo for his helpful comments and our colleagues, particularly Mr. Shigeru Oyanagi at Toshiba Research and Development Center, for their valuable discussions and various supports.

REFERENCES

1. Bochmann, G. V. and J. Gecsei, "A Unified Method for the Specification and Verification of Protocols," *Information Processing 77,* North-Holland Publishing Co., 1977.
2. Enslow, P., "Distributed Data Processing Systems: Some Definitional and Research Issues," Distributed Processing Workshop, Brown University, August 3-5, 1977.
3. Jensen, D. E., "Distributed Processing in a Real-Time Environment," Infotech State of the Ant Report on Distributed Systems, 1976.
4. Jensen, D. E., "The Honeywell Experimental Distributed Processor—An Overview of its Objectives, Concepts, and Facilities," Distributed Processing Workshop, Brown University, August 3-5, 1977.
5. Koenigsberg, E., "On Jockeying in Queues," *Management Science,* 12, 3, Jan. 1966, pp. 412-436.
6. Labetoulle, J., E. Manning, and R. W. Peebles, "A Homogeneous Network for Data Sharing: Modelling and Analysis," *Computer Networks,* Vol. 1, No. 4, 1977.
7. LeLann, G., "Distributed Systems—Towards a Formal Approach," *Information Processing 77,* North-Holland Publishing Co., 1977.
8. Maekawa, M., "Queuing Models for Computer Systems Connected by a Communication Line," *JACM,* 24, 4, Oct. 1977.
9. Manning, E. and R. Peebles, "A Computer Architecture for Large (Distributed) Data, Bases," *Proc. Very Large Data Bases Conference,* 1974, pp. 405-427.

Practical problems in a distributed application

by ERIC D. CARLSON and MARY C. SMYLY
IBM Research Laboratory
San Jose, California

INTRODUCTION

In this report the term "distributed application" is used to refer to application programs with modules which operate on two or more computers, and which communicate with each other in real time. In particular, this report describes experience with an IBM Program Product, Trend Analysis/370 (TA/370). TA/370 is a distributed application employing two computers, a host and a satellite. The satellite computer is used for a color graphics feature, which can produce line graphs or bar charts in multiple colors. The host computer is used for data base management and non-graphical data transformations (such as computing ratios). Figure 1 shows the TA/370 system configuration, and details on the system can be found in the reference manuals.[5,6] In TA/370, user queries are processed in the host computer. If the query contains a request for a graph or bar chart, the appropriate parameters (data and formatting) are transmitted from the host to the satellite. The satellite acts as a formatter and controller for the graphics terminal on which the graphs and bar charts are displayed. Thus, only the graphics formatting and control functions in TA/370 are distributed, and communication is from the host to the satellite.

Of the many possible reasons for a distributed application, the reason for distributing the graphics functions in TA/370 appears to be performance. Specialized graphics formatting functions, such as labelling and scaling, can be performed in the satellite based on encoded commands from the host. Thus, not all the formatting data for a graph need be transmitted from the host to the satellite (reducing transmission time) and the formatting commands can be executed in the satellite. Reduced transmission times and a special-purpose graphics formatter are intended to improve response time and reduce costs for creating graphical outputs in TA/370. To achieve these benefits, a special-purpose protocol for data communications was developed, and the satellite programs for graphics formatting were application-specific and coded in assembler language.

Our experience with TA/370 began with an attempt to modify the code for the satellite in order to be able to use different i/o hardware than supported by TA/370. These were differences in implementation and not in functional capability. Because of the special-purpose nature of the satellite code and data communications protocol, we found modification difficult. Attempting to add new graphical functions to TA/370 appeared even more complicated. The difficulty in modifying and extending the graphical capabilities of TA/370 led us to investigate the possibility of using a general-purpose graphical subroutine package, developed by IBM Research for other applications,[2] in TA/370. Because this subroutine package was developed to operate on the host computer, we expected that we would lose the performance advantages of the distributed processing for graphics in TA/370.

We implemented a prototype revision of TA/370 using the host-based graphics subroutine package. In comparing the prototype with the original TA/370, there was no noticeable change in response times. Further investigation of the two versions was made to find out why the theoretical performance advantage for distributed processing was not realized in practice. The results of this investigation indicated that the overhead in the host-satellite protocol was so large that advantages from reduced data transmission or parallel processing did not affect response time significantly. In addition, advantages from distributed processing appeared to be offset by need for data encoding and decoding to communicate between the host and the satellite. Use of a general-purpose software package operating on the host computer seemed to offer greater flexibility, easier maintenance and extensions, and reduced hardware and programming costs without any performance penalty compared to the original version of TA/370.

In the next section of the report we describe in more detail why we wanted to modify TA/370 and the problems we encountered in doing so. The third section describes the implementation of the "non-distributed" prototype and the benchmark comparison with the distributed TA/370. In the fourth section we analyze the results of the comparison, and in the fifth section we discuss some alternatives for distributed processing in applications such as TA/370.

THE PROBLEMS WITH THE DISTRIBUTED VERSION

We were interested in modifying TA/370 for two reasons: (1) to make it operational on a satellite computer with a different graphics terminal and host-satellite communications interface, and (2) to add new graphical functions (such

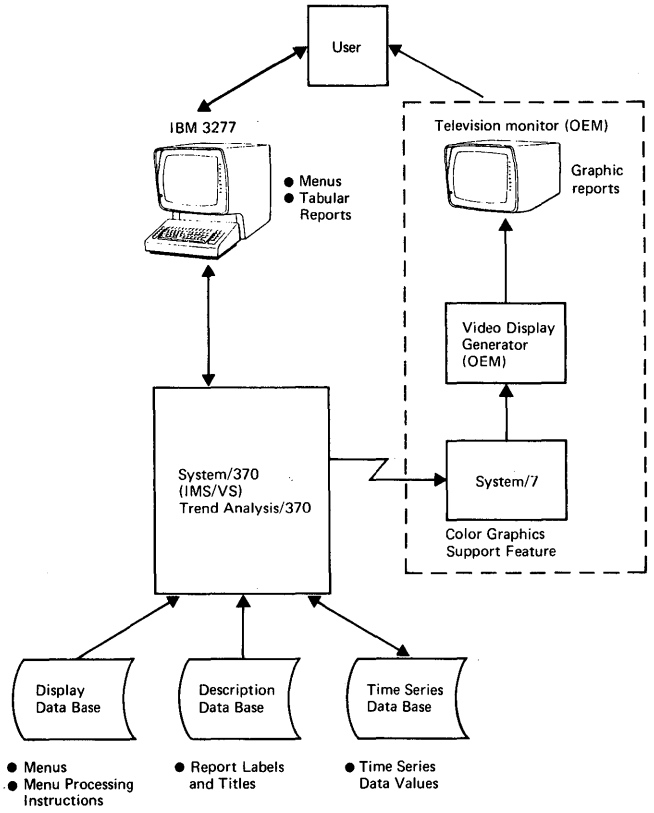


Figure 1—Trend analysis/370 configuration

as drawing maps). The modification to support different I/O hardware involved adding a few hundred lines of code to the TA/370 modules in the satellite. The addition of new function would require adding modules in both the satellite and the host. Because of time constraints and the difficulty of modifications, only the modifications needed to support the I/O hardware were completed.

In making the modifications we encountered several problems:

1. The satellite code was special-purpose, designed for this application and satellite computer, and was written in assembler language. Even after six months of work, we never totally understood the satellite code, and the programmer who wrote the original code had to make the modifications. Even if we had been experts in the assembler language for the satellite, modifying the code would have been difficult because it required detailed knowledge of the hardware and of the data communications protocols.
2. Communication between the host and the satellite used a special-purpose set of records, with five basic record types, each of which was encoded at the bit level. That is, instead of using an extension of a standard communications format, such as the ASCII character set, TA/370 essentially used five special-purpose bit strings for which we had to learn the encoding and decoding.
3. The distribution of graphics function had resulted in duplication of functions in the host and the satellite computers. Specifically, the host modules scaled the data to be displayed into value ranges compatible with the range of values which could be represented in one word on the satellite, and the satellite rescaled the data into ranges compatible with the address space of the graphics display. In addition, the host modules encoded the graphics commands and data into the five special purpose records for transmission to the satellite, and the satellite modules decoded these records and then recoded the data for transmission to the graphics terminal. Thus to make any modifications to the graphics modules, one had to understand the details of both the host and the satellite coding schemes.

In looking at more complicated modifications of TA/370, such as adding new graphics functions or supporting a different type of graphics terminal, it was apparent to us that problems, such as those listed above, would become more severe. We would have to add and modify code in both the host and the satellite and we would have to modify or extend the data communications records.

EXPERIENCE WITH A NONDISTRIBUTED VERSION

Graphics functions in the nondistributed version

We had several years experience with an application similar to TA/370 in which the graphics functions were not distributed.[2] This system used a general-purpose graphics subroutine package, called DISPLIB, which was developed to support a variety of graphics functions and graphics terminals. This package, written in FORTRAN, provides basic commands for displaying text, lines, and points, as well as for display transformation (e.g., scaling) and user input. The package is similar in style to the standard subroutine package currently being proposed by ACM SIGGRAPH.[1] The package also uses extensions of the ASCII or Correspondence character sets for graphics communication to the terminal; this convention is used by several terminal manufacturers. Because the package operated on the host computer, its use in TA/370 required that we not distribute the graphics functions. We expected that this change to a nondistributed application might degrade performance.

A prototype version of TA/370 using DISPLIB was implemented. The prototype supported only simple bar charts and therefore was not functionally equivalent to the original TA/370. In addition, the prototype simply translated the five records (that TA/370 created to send to the satellite) into DISPLIB calls. It would have been more efficient to replace the TA/370 code that creates the five records with code calling DISPLIB, but simply translating the records required fewer changes to TA/370 and was easier to program.

Comparison: distributed and nondistributed

Table I gives a comparison of the graphics functions in TA/370 and those in our prototype revision of TA/370. This comparison indicates the basic differences between the two versions.

The prototype was implemented and a benchmark bar chart containing data on six organizations for five years

TABLE I.—Comparison of Graphics in TA/370 and TA/370 Prototype

Feature	TA/370	TA/370 Prototype
graphics package language	assembler	FORTRAN
style	special-purpose	general-purpose
graphics function implemented on	satellite	host
graphics functions invoked by	building graphics records	subroutine calls
host code language	PL/1	PL/1
type of graphics	line graphs and bar charts	simple bar charts
communications	5 special record types	ASCII extension
what communicated	data + graphics format codes	data + graphics formats
terminals per satellite	4	1
terminal types supported	1 (raster)	3 (raster, vector, storage)
size of host code for graphics (approximate)	29K bytes	113K bytes
lines of host code written by applications programmer (approx.)	500	170
satellite code size (approximate)	47K bytes	24K bytes

(=30 bars) was used to compare performance. Amount of code and communication load also were compared, and flexibility and programming cost were compared subjectively.

1. *Response time:* For each version the time to draw the bar chart was about 15 seconds. (Note: in other installations of TA/370 which use a S/7 with hardware features to assist in host-satellite communications, response times for bar charts similar to the one used in our benchmark are about 10 seconds.)
2. *Communication load:* In the original TA/370 implementation about 800 bytes of bar chart values and formatting keys were transmitted from the host to the satellite. In the prototype version of TA/370 about 2500 bytes of bar chart values and formatting data had to be transmitted. Transmission was at 1200 Baud. TA/370 sends 268 byte blocks and the prototype used 256 byte blocks.
3. *Communications protocol:* Both versions use 1200 Baud asynchronous communication over voice grade lines. The original version uses a transmission code (Paper Tape Transmission Code) developed for IBM 2740-1 terminals. The prototype based on DISPLIB uses ASCII or Correspondence codes. The protocol in the original version uses five special-purpose data communications records with host-satellite "handshaking" after each block is transmitted. The prototype uses "standard" ASCII codes with "handshaking" only at the start of host-satellite communications.
4. *Amount of code:* Even though we implemented only a subset of the TA/370 graphics functions, the prototype version of TA/370 required much more code in the host because a general-purpose package was used. However, the applications programmer wrote less code in the prototype because this package was available. The code written by the applications programmer in the prototype is about 10 percent of the total amount of code, whereas in the original TA/370 it is 100 percent. We estimate that if we had chosen to implement only the TA/370 graphics functions (eliminating any DISPLIB code not required for these functions) and to rewrite the TA/370 host code for (creating graphics records rather than adding code to translate those records into DISPLIB calls), the amount of host code in both versions would have been approximately equal. Any implementation of the prototype would require less code in the satellite. Moreover, the satellite code used in the prototype would support all the TA/370 graphics functions and would not require modifications for most additional graphics functions.
5. *Size of satellite computer:* The original TA/370 required an IBM System/7 with disk storage, while the prototype could operate using an IBM 5100. The System/7 is about five times more expensive that the 5100 (about $50,000 more).
6. *Programming cost and skill level:* Because the prototype used a high-level language of graphics calls rather than having to create records at the bit level, we believe that the cost to program the prototype was less than the cost to program the equivalent function in the original TA/370. The prototype was developed in less than two person months for a programmer who was not familiar with the DISPLIB package. In addition, both the host and satellite code for graphics in the prototype would not need to be changed if new graphics functions were required. This type of general-purpose graphics

would greatly reduce the programming cost for modifications of TA/370.

7. *Flexibility:* Because the prototype used a high-level language for graphics functions, and a general-purpose graphics subroutine package which supported several types of terminals, we believe that the prototype version was more flexible and would be easier to modify or extend than the original version of TA/370.

In summary, the experience with the prototype indicated that a nondistributed version of TA/370, using a general-purpose graphics subroutine package would not degrade performance, even though the communication load was increased, and would offer significant advantages in terms of hardware costs, programming costs, and flexibility. The performance results of the benchmark were not as expected, so the prototype and original versions were further analyzed to try to better understand the performance implications of the distributed versus the nondistributed implementations.

ANALYSIS: DISTRIBUTED VERSUS NONDISTRIBUTED VERSIONS

One could argue that the prototype performed as well as the original version of TA/370 because we implemented only a subset of the graphics functions of TA/370 and because we had the advantage having the original version as a model for implementing the prototype. Both these arguments are partially true, but further analysis indicated that even with equal functions and equal skill in implementation one should not expect much difference in performance between the distributed and nondistributed versions. The analysis indicated that the predominate factor in performance was the time to transmit data from the host to the satellite over the 1200 Baud line. For the bar chart used in the benchmark comparison the original version of TA/370 transmitted less (about one-third as much) data than the prototype version. However, in the original version of TA/370 there is "handshaking" between the host and the satellite once for every transmitted block. This "handshaking" is implemented in software. The "handshaking" consumes enough time to eliminate any performance advantage for the original TA/370 due to reduced data transmission.

This informal analysis of the performance of each version is substantiated by an analysis using a simple model. This model is a simplification of a model of distributed (graphics) systems proposed by Foley.[3] The model consists of four factors which affect performance: data base access time in the host, execution time in the satellite, and data transmission time. Thus for any graphics function in either version, a model of performance is:

T=TD*ND+TH*IH+TS*IS+TT*TL

where:

T=total time (e.g., response time)

TD=average time for data access in host, ND=number of data base accesses

TH=execution speed of host, TS=execution speed of satellite

IH=instructions executed in host, IS=instructions executed in satellite

TT=transmission speed, TL=transmission load

For either the distributed or nondistributed versions of TA/370, the TT*TL term in the model is the predominate factor. Transmission speed is 1200 Baud and the transmission load is in thousands of bits, so that TT*TL results in times measured in seconds. The data base access times (TD*ND) in both versions are equal because the same data must be retrieved. These times are usually in hundredths of a second. The host and satellite execution speeds are approximately equal because the satellite is about one-third as fast as the host, but the host is time-shared and time-sharing, on the average, reduces the effective execution speed by about one-third. The execution plus data base access times (TD*ND+TH*IH+TS*IS) in both versions of the system are in tenths of seconds. Thus, the model indicates that any performance difference between the two versions is going to be due to differences in transmission times. Detailed expressions for computing the transmission loads in each version were worked out, but are omitted here. In general, for a simple graph or bar chart the transmission load in the original TA/370 is about half that in the prototype (approximately 600 bytes compared to 1200 bytes). For a complicated graph it may be only one-sixth as much (about 1000 bytes compared to 6000 bytes). Note that at 1200 Baud the transmission load in the prototype version could take up to 40 seconds to transmit. In practice, however, the performance advantage that this simple model predicts for the original version is not realized. As previously mentioned, the practical failure to realize this advantage seems to be due to the overhead of the transmission protocol used in the original version.

ALTERNATIVES FOR DISTRIBUTED FUNCTION

Models of distributed applications

To choose among alternative configurations for a distributed application such as TA/370, one could use the model proposed by Foley.[3] Given a set of alternatives for host, satellite, and data link hardware this model can be used to select the configuration which will produce the best response time for a given cost constraint. However, one needs to have predetermined the distribution of function and to have estimated the frequencies of graphic macro instructions, the probabilities of various user interactions, and the data base accesses required per interaction.

It would be very difficult to estimate the parameters of TA/370 and the parameters of hardware alternatives needed for the Foley model. More significantly, the hardware alternatives for TA/370 are restricted by the application's requirement for a color graphics terminal which could be attached to a host over voice grade lines. Therefore, the Foley

model does not provide much help in selecting among alternative distributions of function for TA/370. However, Foley used his model to analyze four "typical" graphics applications, one of which, two dimensional drawing, is similar to TA/370.[3] Foley's analysis considered over 12,000 possible hardware configurations. Based on this analysis, Foley recommended that, in general, the best system configuration for graphics applications consisted of the simplest possible satellite (which provided enough function) using a voice-grade (2400 Baud) data link. Foley's analysis indicated that the data link and bulk storage on the satellite are the two components which most influence response time. Foley's analysis of distributed applications thus is very similar to our study of TA/370: data transmission load is the key variable affecting cost/performance in a distributed application, and distribution of function (or data) is useful only to the extent that it helps reduce this load.

Foley's model gives us insight in selecting a hardware configuration, given a set of functional modules for a distributed application. For a given hardware configuration and function modularization, a graph partitioning model, such as that suggested by Hamlin,[4] can be used to decide which modules should be distributed. In a graph model of a program, the modules are nodes and among the nodes there are directed arcs which represent the calling and returning structure of the program. Associated with each arc is a cost, such as the amount of data passed or the overhead time for calling. Other costs, such as the amount of global data accessed from each module or the execution time for each module on the host and on the satellite, may be associated with each node. The graph partitioning algorithm attempts to minimize the cost associated with assigning the modules (nodes) to the host or the satellite. To do this, the model requires an estimate of the frequencies of data accesses and inter-module calls. For an example of such a model see Hamlin.[4] As indicated by Hamlin's example, unless there are constraints (such as limited memory space) which prevent all modules from being assigned to one computer, the cost function usually is minimized if there is no distribution of function.

Note that both the Foley and Hamlin models assume that the program modules are defined. Yet the definition of the modules is a primary factor affecting the cost functions in both models. This suggests the need for a model based on the costs of performing various graphics functions, independent of program modularization, which could be used to analyze alternative modularizations. In addition, both models require many parameters of the application, program, and hardware. Estimating these parameters is difficult, and likely to be error prone. To the extent that the models are sensitive to errors in the parameters one would expect the models to suggest suboptimal alternatives which, if used, would need performance tuning. For performance tuning, it would be useful to be able to move program modules between the host and satellite computers. To make this easier, one needs to be able to compile and execute the same language on both machines,[4] and to have a mechanism which supports calls and parameter passing among modules on different machines.

Possible alternatives for TA/370

The general guidelines of the Foley and Hamlin models for distributed applications suggest two alternatives for TA/370.

1. Reduce the size of the satellite computer and the modules assigned to it to as small as possible to contain the bar chart and graph formatting functions. Place all remaining modules in the host. This distribution will limit data transmission to data values only. The double encoding/decoding and data scaling currently in TA/370 will be eliminated. Our prototype is representative of this alternative.
2. Distribute the entire application program to the satellite, with batch transmission of a subset of the data base to each satellite on an as-needed basis. However, both the data base and the application program are too large for most satellites. A variation of this alternative would be to keep a working-set of the data base in the satellite (e.g., the data used in the previous "*n*" displays), and distribute all the graphics functions to the satellite, as done in the original TA/370. This alternative will reduce the data transmission because all the data for a graph or chart will not need to be transmitted if it is a variation on one of the charts stored in the working-set.

For either alternative it would be beneficial to use the highest speed communication link compatible with the applications' requirement for a voice-grade line. Currently 4800 Baud is possible, although to use 4800 Baud would require changes to the satellite hardware (System 7 or 5100) and to the communications protocol (bisynchronous instead of asynchronous).

CONCLUSIONS

The experience with TA/370 identified several practical problems in distributed applications:

1. The potential performance advantage of distributed function was reduced because of the overhead of host-satellite communication protocol.
2. Distributed function resulted in special-purpose programming and data encodings which made modifications and extensions difficult.
3. Distributed function greatly increased the cost of the satellite computer, without compensating savings in the cost of the host computer.
4. Distributed function probably increased the amount of code and the programming cost for this application.
5. Transmission time between the host and the satellite was the primary performance factor. The application's need for a 1200 Baud line limited the performance improvement that could be achieved by distributed function as long as the data base was stored on the host.

6. Deciding how to distribute function involves considering hardware configuration, alternative distribution of program modules and data, and alternative modularizations. Existing analytic models of distributed applications do not encompass all of these considerations.
7. In designing distributed applications such as TA/370 the constraints are: the available hardware, the hardware costs, and the software development and maintenance costs. As with most programs, the software development and maintenance costs are likely to be the most important constraint.

This study does not indicate that distributed processing is ineffective for applications such as TA/370. It does indicate that distributed processing is not simply a separation of function among computers, because the separation requires communication. Because communication is involved, design decisions can become more complicated (as indicated by Foley's model), and unexpected inefficiencies can result (such as the "handshaking" problem). Most importantly, when the design objectives change (e.g., added function or different hardware), a distributed implementation may be more difficult to modify than a non-distributed version. Our experience with TA/370 indicates that these "practical" problems can eliminate a theoretical advantage for a distributed application.

REFERENCES

1. Association for Computing Machinery, Status report of the graphics standards planning committee of ACM/SIGGRAPH, *Computer Graphics* 11:3, Fall 1977.
2. Carlson, E. D., et al., "The Design and Evaluation of an Interactive Geo-Data Analysis and Display System," *Information Processing 74,* North Holland Publishing Company, Amsterdam, 1974, pp. 1057-1061.
3. Foley, J. D., "An Approach to the Optimum Design of Computer Graphics Systems," *Communications of the ACM,* 14:6, June 1971, pp. 380-390.
4. Hamlin, G., "Configurable Applications for Satellite Graphics," Ph.D. Dissertation, Department of Computer Science, University of North Carolina at Chapel Hill, 1975.
5. IBM Corporation, *Trend Analysis/370 General Information Manual,* GH20-1861, IBM Corporation, Data Processing Division, White Plains, New York.
6. IBM Corporation, *Trend Analysis/370 Program Reference Manual,* SH20-1937, IBM Corporation, Data Processing Division, White Plains, New York.

A panel session—Distributed systems—A challenge to management

SESSION CHAIRMAN—GEORGE M. CRANDELL, JR.
McKinsey & Company, Inc.

Panel Members

Lester Stubbs—Mattel Toys
Michael C. Dowling—Fireman's Fund Insurance Company
Mario Calderin—Aratex, Inc.

PANEL OVERVIEW—George M. Crandell, Jr.

This session addresses the management issues and decisions that concern MIS directors, project leaders, and users in the implementation of distributed systems. The panelists, who have been selected to provide a cross section of industries, will briefly describe their distributed systems and then focus on the most significant managerial/organizational challenges encountered during implementation. The pros and cons of alternative approaches will be evaluated, and mistakes as well as successes will be discussed.

Areas to be explored include:

1. Considerations and caveats in allocating responsibilities between headquarters and local MIS personnel, users, hardware vendors, and software houses/consultants
2. Approaches for carrying out system design, development, implementation, training, maintenance and trouble handling
3. Lessons learned by the panelists from their experiences, and their suggestions for avoiding potential pitfalls.

There will be considerable time for questions and general discussion following the presentation.

INSTALLING IBM 3790's IN A MANUFACTURING ENVIRONMENT—Lester Stubbs

During the last two years Mattel Toys has installed various phases of its new manufacturing system utilizing the IBM 3790 in a distributed processing environment. Each installation has been extremely smooth and relatively error-free. Besides having luck on its side, Mattel used a project management approach throughout the development and implementation process that contributed greatly to this success. This presentation is centered around the actual implementations and the planning that went into them.

There are two extremely important factors that must be continually evaluated for any new system being developed. The first is constant review to make sure that the system will meet the end-users' needs and that the end-users can effectively work with the system in a production environment. The second factor is selecting and tailoring the equipment so that it will perform properly when in production. A closer look at how Mattel approached these two factors will be presented.

As installation nears, there is system testing, user testing, acceptance testing, system documentation, equipment installation and conversion. How Mattel approached each of these tasks will be reviewed. The project management approach used will be emphasized and the critical factors will be evaluated.

Finally, there is the day of installation. If properly planned, this event can be very anti-climactic. The first few weeks of production will be reviewed with emphasis on Mattel's warranty period concept.

Everyone hopes to install a system that is a glowing success, and, through personal experiences and the experiences of others, system installations can result in rewarding experiences for all. Hopefully, this presentation, and others like it, will help.

DISTRIBUTED DATA PROCESSING IN A CLERICAL ENVIRONMENT—Michael C. Dowling

Fireman's Fund direction and plans in distributed data processing must be taken in the context of the nature of its business, its field organization, its current data network, its relationship to American Express Data Processing, and its view of the business process and its role in the marketplace over the next 10 years. Given our structure and business needs we sought a means of reducing our operating expenses while improving the quality and speed of services we provide, i.e., insurance policies, quotes, and loss payments. Data processing and improved manual procedures provided a basis for achieving this business objective.

In the data processing area, we looked at the technology

available and found that three fundamentally different approaches were available:

1. Batch store and forward
2. On-line processing
3. Local processing.

Our research into the business needs indicated that to reduce cost and improve service we should use all three approaches in an optimum mix. Thus we are aggressively pursuing the development of systems utilizing all three fundamental approaches. From a strategic point of view, we intend to place computing power in the hands of our local offices while retaining full control of programming and program version release at the central site.

In order to adequately address the new business and systems environment of remote computers and computer processing in essentially clerical offices, we established a Distributed Systems Department. This department is responsible for all local applications and the local portion of batch store and forward applications. The department reports to the Senior Systems Executive and is essentially a microcosm of the total systems organization.

Not only did the systems organization undergo substantial revision, but new user departments were created and an extremely active and important executive steering committee was established. The business executive was made responsible for priority setting, business definition, systems justification and establishment of a responsive Business Systems Plan.

As our systems become an integral part of our daily field operations, we are learning what manufacturing companies with computer assisted production have known for some time: that a new level of responsiveness to system problems is required. We have been deeply committed to this task for the last two years and we have learned lessons in every area from equipment selection to field implementation. Most importantly, however, we have developed from our experience seven postulates for effective distributed system development:

1. Demand user participation
2. Insure management commitment
3. Maintain high visibility
4. Fund development on a step by step basis
5. Define accountability
6. Balance technology with business needs
7. Integrate manual systems planning with EDP systems planning.

DISTRIBUTED SYSTEMS IN RETROSPECT: LESSONS WE LEARNED THE HARD WAY—Mario Calderin

Aratex, Inc., an Encino, California based textile rental company, installed one of the first 3790 distributed processing networks in the country. Running under the Systems Network Architecture (SNA), the 3790 has brought new data processing capabilities to the company's remote locations.

At each of Aratex's 35 locations across the country, plant personnel using 3271 display terminals linked to a 3791 controller, which acts like a small computer with its own locally stored data, constantly update and interrogate data regarding Customer Billing, Accounts Receivable, Payroll, Sales Analysis, and Inventory Control. All data is checked for validity using extensive edit programs. Since the system is installed at locations where there is no EDP expertise, all transactions are checkpointed to provide automatic restart capability.

At the end of the day, all pertinent data is transmitted unattended to a 370/148 at the Encino National Data Center. After the necessary processing is performed, approximately 500,000 lines of output is transmitted back to the remote locations daily, where it is printed on the 3790 printer. A 3790 system is also installed as part of the network at the company's Central Warehouse facility in Fresno, California. The system accepts on-line order entry transactions from 3790's at any plant in the network and produces order status, packing slips, shipping confirmation, and invoices at the warehouse, as well as assigning goods for replenishing random locations, and regulating the warehouse work schedule based on order arrival frequency.

The entire system is comprised of over 400 programs developed over the last two years. As one of the first companies in the United States to deal extensively with distributed processing, we have developed expertise in the control aspects. Our experience indicates that the most important areas to consider are:

1. Control of programs at remote sites
2. Data synchronization between remote sites and central data bases
3. Communication programs to allow simultaneous transmissions to and from the host system by multiple users
4. User backup and recovery in case of down-time
5. Response to user questions while operating the system
6. Centralized problem determination for both hardware and software
7. User training
8. System installation guidelines.

Area Director
Charles W. Bachman
Honeywell Information Systems
Billerica, Massachusetts

Database management systems

Database management as a technology has made tremendous strides in recent years. Database management systems are offered by every major hardware vendor and many software houses. Installations which operate database management systems in support of their management's objects now number in the thousands. Two database management specialized conferences are held each year with hundreds of attendees present. The first of these is held by the ACM Special Interest Group on the Management of Data (SIGMOD). The second is the Very Large Database Conference (VLDB) which has moved around the world with Germany as the host for 1978. These newer conferences have collected most of the leading edge database technology papers because it is possible to concentrate more of the research-oriented people, their papers and discussions into smaller towns and smaller hotels.

Therefore, the Database Management sessions for NCC '78 have focussed upon the problems and solutions associated with use of the currently available database management systems. Four panel discussions and four technical sessions have been planned for your attendance at NCC '78. Three of these panel sessions are focussed upon users' experiences relating to their database systems. One panel session focusses on "end user facilities," one of the leading edge areas of database management. The four technical sessions bring papers which are oriented toward improving performance of database systems, distributed database systems, conversion and language aspects of database systems.

For the many thousands who may not be familiar with the concepts of database management systems, a short primer may be appropriate so that the panel discussions and technical sessions will make some sense.

What is database management? Roughly, this means the establishment, and maintenance of computerized collections of information about an enterprise (manufacturing company, bank, hospital, school or other organizations). This infor-

mation, stored in permanent files, contains historical, current and future planning information about the enterprise. All this information is collected and preserved at considerable expense so that it will be available when needed; to answer a question, to assist in making a decision, or to guide someone as to the next task to execute. This collection of information has been called the database, as it forms the basis or foundation upon which the operations of an enterprise is built.

Database data represents one of the two kinds of data found in a computerized information system. The other kind is message data. The easiest way to understand the difference between the two kinds of data is to ask the question "what happens after the data has been processed?" Database data is characterized as being put away and remaining static until some process subsequently requests its retrieval. Message data, on the other hand, is active. It is being transferred by one process because it is relevant to the actions of another process which is anticipating its receipt. One or both of these processes may be manual and outside of the computer. Thus, the message data and database data complement each other and represent the "hurry up" and "wait" of the computerized information systems. Database Management is the title given to the art of storing and retrieving data. Message Management and Communications Management are the titles given to logical and physical aspects of moving data between processes. The reader is urged to look at the sessions with titles using the term "communications" or "networks" or "distributed systems" to find additional material on the current state of the data movement technologies.

Network database evaluation using analytical modeling*

by TOBY J. TEOREY and LEWIS B. OBERLANDER
The University of Michigan
Ann Arbor, Michigan

INTRODUCTION

The proliferation of computerized databases and the widespread demand for timely access to data has resulted in the need to develop automated techniques for database design. In the past, database design was accomplished manually by data processing personnel through past experience and trial and error. However, the implementation of large integrated databases in organizations made it increasingly difficult for individual users to do the necessary design. A new approach was required to consider the many tradeoffs among user objectives and within the complex logical structures required to integrate the various types of data.

The stages of database design have only recently been well-defined,[5] and much of that work is still done manually with some assorted design aids available at some of the stages. One of the more quantifiable stages in the database design process is the implementation of a physical database from a logical database design. Logical database design results in the definition of record types and their relationships under the constraints of a particular data model, but independent of physical structure. The design of the physical database must take into account the variations in access methods, i.e., the access paths and search mechanisms available, as well as physical clustering of records near each other within blocks or in a contiguously defined set of blocks. Considerably more insight is required in physical database design before determining whether or not it can be accomplished independently of logical database design.

Currently there exist some very general physical database designers[10,13] which address tradeoffs among major classes of storage structure. On the other hand, several evaluators of physical databases have also been proposed. Some are simulation models and are quite expensive to run.[2,7,9] Others are probabilistic and are limited by expected value assumptions.[3,11]

This paper described the concepts leading to the development of an operational Database Design Evaluator (DBDE), which accounts for the more significant parameters affecting physical database design performance and yet is computationally fast enough for consideration as the central module of a physical designer. The DBDE is a software package which evaluates Honeywell's Integrated Data Store or IDS.[6] While the model produces "expected value" results, it goes beyond the usual expected value assumptions and considers the effects of the distribution of logical record sizes and placement. The DBDE is directly applicable to CODASYL database systems.

The most easily identifiable forerunner to the DBDE was the general analytical method of evaluating physical database designs for single record type files proposed by Yao.[13,14] The general method decomposed all candidate organizations and access methods into several basic parameters: the number of index levels, the expected search length at each index level and the level of usable data, and the fraction of sequential accesses versus pointer (random) accesses at each level. All applications were assumed to consist of randomly addressed operations such as queries and updates.

The File Design Analyzer[11] extended the basic method to consider both sequential and batched processing in addition to random processing, and it extended the concept of the pointer (random) access parameter to specify the degree of randomness required by each configuration. An example of this is the implementation of overflow in an indexed sequential organization. The pointer to an overflow record could refer to another point on the same track, another track with the same cylinder, a different cylinder or even a different device. Each specification would imply a different expected access time to the overflow area. This extension enabled the File Design Analyzer to determine a range of performance between "single access" (i.e., a dedicated device with no seek time, but including an average rotational delay between consecutive sequential accesses) and "multi-access" (i.e., a shared device with the worst case conditions of all consecutive sequential accesses becoming random accesses in reality). The purpose of this dichotomy was to place bounds on I/O service time for the extremes of one user and many users. No attempt was made to determine the effect on queuing delays with this model or with the DBDE.

The DBDE does, however, represent significant improvements over the previous models in the following areas:

1. Computation of database overflow probabilities on the basis of database growth and distribution of record size, and determination of the effect of increasing

* Supported by the Defense Communications Agency, Contract Number DCA 100-75-C-0064 for the WWMCCS ADP Directorate, Reston, Virginia

overflow on the I/O service time required to execute various data manipulation operations.
2. The effect of buffering on I/O service time.
3. The modeling of "currency" and dependent sequences of operations performed on network databases.

Other analytical models have been confined to storage structures for flat files or hierarchical systems, but have not considered network databases. The proliferation of CODASYL DBTG or similar implementations for network databases have increased the need for network evaluation models.

INTEGRATED DATA STORE (IDS)

The Integrated Data Store (IDS) is a host language Database Management System which interfaces extensively with COBOL. It was developed in 1963 by General Electric and is now being used on the Honeywell 6000 series and other configurations. It is widely regarded as a forerunner of the CODASYL DBTG concepts, and is very close to the specifications of the 1971 DBTG report.[4] It was developed to organize information so as to minimize redundancy and duplication of records or data, provide a single database for many applications, store and retrieve logically related data in cohesive manner, and provide an efficient query capability.[6]

The structure of IDS may best be summarized in terms of the interrelationships between records and access mechanisms, and chain classification.

Record classification

A record may either be a master record or a detail record. Masters and details are related to each other by means of chains. A chain may contain only one master record. However, a record may be the master or detail of more than one chain, and a record may be a master in one chain and a detail in another chain simultaneously. All IDS records of a specific record type are fixed length and fixed format. Different record types may have different lengths and formats.

Another form of record classification is by the retrieval access mechanism. Three methods are allowed in IDS:

1. Primary records—retrieved by reference codes which point directly to the physical page and line number on which the record resides. The page is the basic unit of transfer between secondary storage and main storage. It is a physical block (with fixed size in subfiles, somewhat analogous to DBTG areas) that may contain many record types simultaneously. PAGE RANGE is a feature of IDS that specifies a set of physically contiguous blocks (pages) to contain records of a given type or types.
2. Calculated records—retrieved by calculating a page address (via hash code) from a particular key value. A sequential search in the page and possibly an overflow page yields the desired record.
3. Secondary records—These are records that can only be retrieved by accessing the master records and then traversing the chain to which they belong. Often it is possible that the immediate master cannot be accessed directly. In this event, master records that are on a higher level than that of the required immediate master need to be accessed. The search is then performed from these higher level records down to the immediate master in question and then finally to the desired secondary record. Such higher level masters are known as entry points and are accessed via primary record operation, calculated (CALC) record operation or by a RETRIEVE master record operation.

Chain classification

In IDS a record may belong to any number of chains. If retrieval of a record is specified by a particular chain, it is retrieved via that chain. If the chain through which the record is to be retrieved is not specifically stated, the record is retrieved via its prime chain which is declared a priori (in the RETRIEVAL VIA CLAUSE). This manner of retrieval by means of a chain applies only to the secondary records.

Finally, a chain may be implemented as any combination of the options chain PRIOR (backward pointer) and chain MASTER (pointer to the master record). Every chain has a NEXT (forward) pointer.

MODEL PARAMETERS

DBDE inputs

Categories of inputs and outputs are summarized in Figure 1.

a. Database Description

The IDS model requires the definition of master records and detail records of chains, the page size and page range of subfiles, and the record lengths and page ranges of all record types (see Table I for an example).

b. Workload Specification

The model identifies applications in terms of specific IDS operations called: RETRIEVE, HEAD (i.e., find master record), STORE, DELETE, MODIFY, and QUERY. An application can be defined by a sequence of related retrieval operations, which traverse the network in various ways, ending with a specified operation on the desired record(s). It is also possible to specify location mode in terms of database entry points and type of access method, and the PLACE NEAR option. PLACE NEAR allows the clustering of detail records near master records they are most frequently processed with.

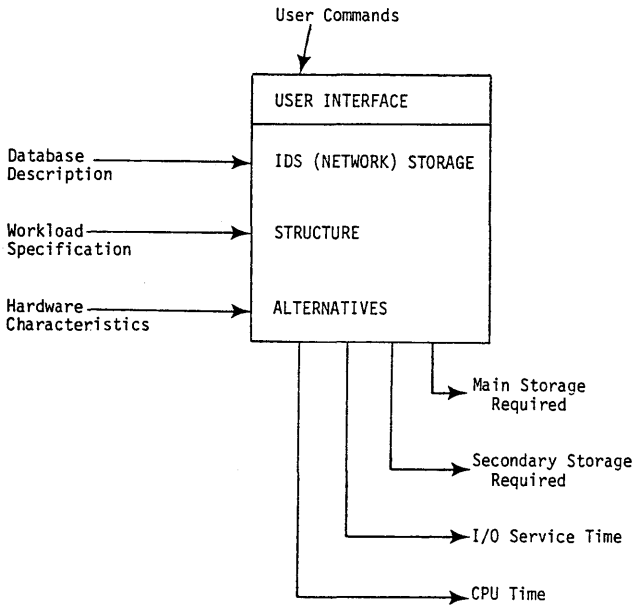


Figure 1—The data base design evaluator (DBDE)

c. Hardware Characteristics

Hardware specifications are, for the most part, limited to the timing characteristics and capacity levels of the secondary storage subsystem. It is assumed that only one type of secondary storage device is used, although the alternative types of devices could be evaluated by reinitializing these parameters. In IDS, allocation across several devices is possible.

Other input parameters which fall loosely under the category of hardware are the estimated CPU time required to initiate I/O starts and completions, the CPU time for moving data within main storage, and the main storage required for support software plus user applications.

One parameter that falls in none of the above categories is the specification that the effect of overflow on performance is desired. The analyst may specify any number of time periods that overflow performance data is to be computed, with the database growth rate implicitly specified by the current database size and the number of record additions per time period. All computations in the DBDE are based on a unit time period. The analyst may choose the most meaningful time unit for his analysis, and merely needs to be consistent with that time unit when specifying workload parameters and overflow time periods.

DBDE outputs

The DBDE outputs reflect an attempt to present a fairly complete picture of resource usage and the components of response time. Enough performance data is given so the analyst can estimate the real cost of the applications evaluated.

a. Main Storage Requirement

This is a straightforward computation, adding the user estimate for object program storage space to the space required for buffers for each application.

b. Secondary Storage Requirement

In IDS this is input explicitly by the user (analyst) in terms of subfile specifications for first and last page numbers.

c. I/O Service Time

I/O time implies elapsed I/O service time, which affects either response time or turnaround time. It differs from channel time in that it always includes all the major components of I/O service: seek time, rotational delay, and data transfer. Bounds on I/O time are provided for the extremes of "single access", and "multi-access" configurations described above.

d. CPU Time

The DBDE estimates the CPU time required for the database functions: overhead to start and complete I/O commands (for physical reads and writes), time for data movement between buffers and user work areas where applicable, database search time, and software write verification. The user CPU time to process logical records is merely echoed from the input data, and is an optional feature.

MAJOR COMPONENTS

An algorithm for database sizing and overflow computation

Given a database with N records of varying size, an algorithm to determine its size needs to answer the following questions:

1. How many physical blocks (pages) will be required if the database is organized sequentially and the ordering results in random placement of records of a given size?
2. How many record occurrences will typically fit into a block?

The first question is of interest in modeling the record distribution or population of sequential and indexed sequential database organizations, which can be computed by the DBDE. The second question is of interest in modeling the placement of chains in IDS.

If all records were of the same size, R words, and the physical block size were B words, then the number of records that could fit into a block would be $\left[\frac{B}{R}\right]^-$ where [expr]$^-$ denotes the "floor function" or "greatest integer equal to or smaller than" the actual value of the

TABLE I—Input data for test network database

HARDWARE SECTION	
Secondary storage device	Honeywell 6060/DSS191
Average seek time	30 millisec
Adjacent cyl. seek time	10 millisec
Avg. rotational latency	8.35 millisec
Full rotation time	16.7 millisec
Bytes per word	6
Transfer rate	1,074,000 byte/sec.
Cylinders per disk pack	404
Tracks per cylinder	19
Bytes per track	11904 bytes
Block gap size	12 bytes
CPU time for I/O start + end	1 millisec
CPU time for core-core move	2.75 microsec/byte

IDS SECTION IDS DATA DIVISION	
Number of buffers (Max. is 256)	50
Write verify ON?	NO
Main storage reqmt. for software	100K bytes
CALC chain retrievals/time unit	0
RETRIEVE EACH operations/time unit	0

SUBFILE DIVISION	SUBFILE 1	SUBFILE 2	SUBFILE 3
First page number	1	101	151
Last page number	100	150	200
Page size	320 words	320 words	320 words
Disk number	1	2	3

TABLE I—(Continued)

RECORD DIVISION	Max. 100 record types							
Record number	1	2	3	4	5	6	7	8
Record length (words)	10	90	60	35	65	25	75	40
Retrieval via	CALC	CHN.6	CALC	CALC	CHN.2	CHN.4	CHN.8	CHN.10
No. occurrences	50	50	50	50	50	50	50	50
First page number	1	1	101	151	151	1	1	101
Last page number	100	100	150	200	200	100	100	150
Workload per time unit:								
a. Retrieve direct	5	5	5	5	5	5	5	5
b. Retrieve record	5	5	5	5	5	5	5	5
c. Retrieve current	5	5	5	5	5	5	5	5
d. Store	15	15	15	15	15	15	15	15
e. Delete	10	10	10	10	10	10	10	10
f. Modify	0	0	0	0	0	0	0	0

CHAIN DIVISION	Max. 100 Chain types									
Chain number	1	2	3	4	5	6	7	8	9	10
Master record	3	4	1	1	2	1	2	1	1	3
Detail record	4	5	5	6	6	2	7	7	8	8
Chain order	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST
Link to next	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Link to prior	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
Link to master	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
Record selection	CURR.	CURR.	CURR.	CURR.	CURR.	CURR.	UNIQ.	CURR.	CURR.	CURR.
Retrv next/$\frac{\text{unit}}{\text{time}}$	0	0	0	0	0	0	0	0	0	0
Retrv.master/$\frac{\text{unit}}{\text{time}}$	0	0	0	0	0	0	0	0	0	0
Head opns./$\frac{\text{unit}}{\text{time}}$	0	0	0	0	0	0	0	0	0	0
Place near master	NO	YES	NO	YES	NO	YES	NO	YES	NO	YES

expression. The resulting database size would be

$$\left\lceil \frac{N}{\left\lceil \frac{B}{R} \right\rceil^-} \right\rceil^+ \quad \text{blocks}$$

where [expr]+ denotes the "ceiling function" or "smallest integer equal to or greater than" the actual value of the expression.

The problem becomes significantly more difficult when there are a variety of record sizes and not much is known about their distribution in the database. The database size is extremely sensitive to record placement when there are very few record sizes, but a large difference in those sizes.

An example will help illustrate the possible extreme values. Suppose there were 1000 records of length 20 words and 1000 records of length 90 words and a constant block size of 100 words. If the ordering specified all the large records must come first and all the small records second, the total size of the database would be:

$$\text{DATABASE SIZE} = \left\lceil \frac{1000}{\left\lceil \frac{100}{90} \right\rceil^-} \right\rceil^+ + \left\lceil \frac{1000}{\left\lceil \frac{100}{20} \right\rceil^-} \right\rceil^+$$

$$= 1000 + 200$$

$$= 1200 \text{ blocks}$$

On the other hand, an ordering which alternated large and small records would require 2000 blocks. Any other ordering of records would produce a database size somewhere between these two extremes. For very large databases this nearly 2-to-1 difference in size will greatly affect the estimated performance for many applications, and consequently record distribution must be represented accurately in the model.

The algorithm assumes that the size of any record in the database is independent of the size of its neighbors. If this assumption is not true for sequential or indexed sequential organizations, then the database size must be computed manually. The IDS model allows dependent record placement to be specified in terms of page ranges. The algorithm then enumerates the possible combinations of record sizes that would overflow a physical block and require the populating of the next block, and it obtains a distribution of unused space in each block. The expected used space in a block, called E(used), is easily computed as the difference between block size B and the expected value for unused space. The total database size is therefore computed as

$$\text{DATABASE SIZE} = \left[\sum_{\substack{\text{ALL} \\ \text{RECORD} \\ \text{TYPES;}}} \text{Record\_size}_j * \text{Number\_of\_occurrences}_j / \text{E(used)} \right]^+$$

where a record type is defined by its length and number of occurrences.

The algorithm takes into account parameters such as percent fill and page overhead associated with specific storage structures in order to compute realistic values for E(used).[12]

As an example of the type of analysis required, let the database consist of 1000 records with an expected record size of 200 words and a block size of 320 words. The actual distribution of record size is:

Record Type	Record Size (Words)	Number of Occurrences
A	100	300
B	200	400
C	300	300

For this simple configuration, a random distribution of record types across the database results in the seven possible arrangements of records in a block shown in Figure 2 occurring with the probability P_1, i = 1, 2, . . . , 7. From the

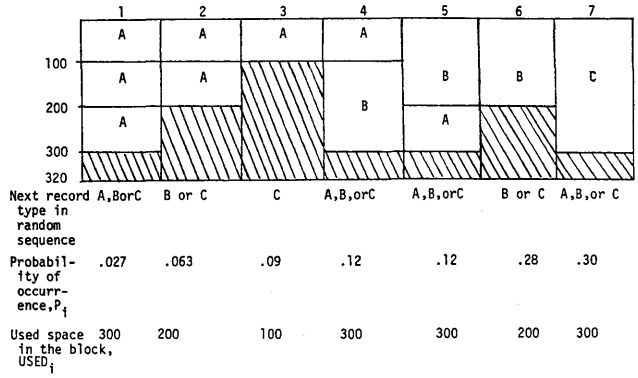


Figure 2—Random sequence distribution of record types in blocks

data in Figure 2 we can compute the expected value of used space in each block and then the database size.

$$\text{E[used]} = \sum_{i=1}^{7} P_i * \text{USED}_i = 247.7 \text{ words}$$

$$\text{Total data volume} = \sum_{j=1}^{3} \text{Record\_size}_j * \text{Number\_of\_occurrences}$$

where j is the number of record types

$$= 100 * 300 + 200 * 400 + 300 * 300$$

$$= 200{,}000 \text{ words}$$

Expected number of blocks required

$$= \left\lceil \frac{200{,}000 \text{ words}}{247.7 \text{ words/block}} \right\rceil^+ = 808 \text{ blocks}$$

The same number of fixed size records of 200 words each would require 1000 blocks. Therefore a significant reduction

is possible in this simple case when the record sizes are distributed over a larger range.

The above computation becomes extremely difficult as the number of record types becomes large (i.e., 500 or more), and even the computer time to exhaustively enumerate all cases is prohibitive. The DBDE uses an algorithm to compute E(used) that reduces the computational complexity to a feasible range for 50-100 record types, which is adequate for most IDS databases. The complete derivation of this algorithm is given in Reference 12.

A natural extension of the database sizing algorithm is to consider new records being added to the database and the probability that they will overflow the page where they would be naturally placed. From this value one can determine the expected number of overflow records generated by a given number of record additions over a particular time period. Overflow in IDS significantly increases the expected retrieval time for records.

Buffering

The original purpose for buffering on second generation (uniprogramming) computer systems was to provide a mechanism for overlapping CPU and I/O time and improve overall performance of each application. Buffering for multiprogramming systems still produces an overlap of resources, but not necessarily for the same application (job). While job A is executing, job B may be inputting into its buffer while job C is done with I/O and waiting for the CPU. The existence of a large number of jobs in the system causes other delays which usually subsume the synchronous CPU-I/O sequences of a sequential data processing operation. Consequently, the benefits of such buffering received by sequential or random database applications in multiprogramming environments are not easily measurable with probability models, such as DBDE, but they can be better approximated by a queuing model for the whole system.

The more directly measurable function of buffering in multiprogramming systems is that of keeping active data in main storage in order to reduce physical I/O operations when data is continually referenced. The DBDE considers this form of buffering, and it allows the analyst to specify up to 256 buffers for IDS applications. Naturally, increasing the number of buffers increases the probability that the next record you need is already in main storage and consequently decreases the expected I/O service time. However, increasing the buffer size too much may result in wasted storage, increased cost for storage used, and main storage allocation delays that may degrade overall performance. Therefore the choice of proper buffer size is critical to an effective IDS application. The DBDE implements the buffer concept by maintaining buffer currency data based on the most recently executed IDS operations and the database definition of chains and record types. It also simulates the basic IDS buffer replacement algorithm, least-recently-used (LRU). Thus, for this part of the IDS model, the DBDE uses a hybrid analytical and deterministic simulation technique.

Currency in a network database

IDS (as well as DBTG databases) has currency indicators which specify the most recently accessed record of a particular record type or master and detail records for a chain type, for example. The DBDE maintains similar information to represent the current status of access to the database. Consequently the next operation is evaluated according to current position and next position desired. The logical structure diagram of the database is implicitly known from the form of the input parameters, and the algorithm consults this information to compute I/O time and CPU overhead to access the next record.

EXAMPLE NETWORK DATABASE EVALUATIONS

The objectives of the test network database evaluations were to validate the IDS model with live test data from an existing database, to provide insight regarding the sensitivity of database performance to the values of storage structure and user workload parameters, and to compare the performance of alternative database designs.

The conclusions from preliminary validation tests ai • that the DBDE is able to describe the major database design parameters and their relationships accurately, that the input data can be created easily and quickly, and that many meaningful experiments can be conducted at a single interactive session with the program. The validation IDS database which represented a large equipment inventory system, was supplied by a client U.S. government agency. The database consisted of 155,000 records of varying length across three major record types, and live test data was collected on an IDS system interfaced with a Honeywell 6060 GCOS operating system for an extensive series of retrieval operations. The test was conducted during peak hours in a large batch multiprogramming environment. Estimates for elapsed time for the live test data ($54*10^3$ seconds) and the DBDE ($42.7*10^3$ seconds) differed by 21 percent. The ability of the DBDE to estimate total elapsed time was due to the availability of an extension which implements a central server queuing model of the test system and estimates total elapsed time based on CPU and I/O requirements derived by the IDS model for the test database. Although these preliminary results have been encouraging, we believe that the validation process should be a continually on-going one as more detailed monitor data is made available. Validation at the individual IDS operation level is currently not possible in the existing environment.

Example 1

The first test database chosen for experimentation was the support database for DBDE,[1] which provided the opportunity to evaluate our own design for DBDE before applying the technology to client systems. The test database schema is illustrated in Figure 3. The database contains eight record types and ten chain types. The database is quite small

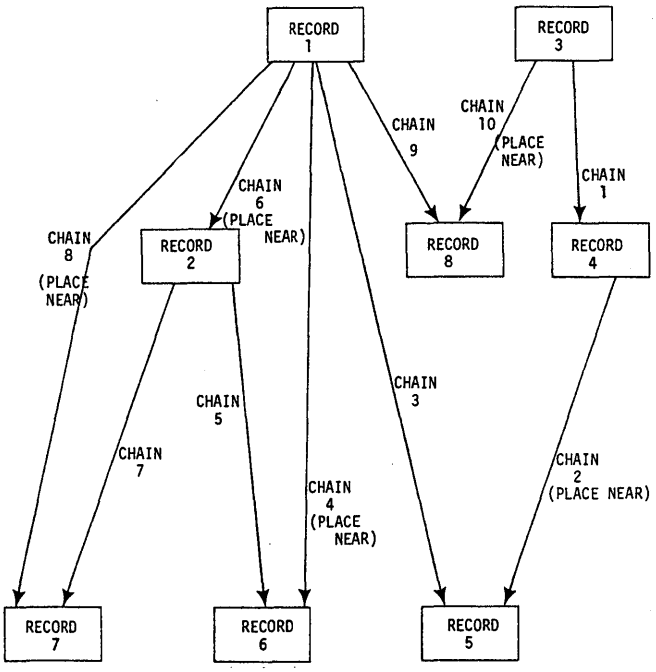


Figure 3—DBDE database schema

(755 records), but this kept the cost down while allowing a great deal of analysis of the model. Other input parameters are summarized in Table 1.

The basic output produced by the program is illustrated in Table II. Each operation is considered separately and totals are produced for given frequencies of occurrence of each operation. Table II illustrates a two-operation application that retrieves a record and modifies it. To accomplish this task, 2 sub-operations are required for each retrieval and 5 sub-operations are necessary for the modify operation. This feature of providing detailed output for each operation enables the analyst to obtain a better picture of cause and effect relationships for overall system performance, and thus make better decisions on parameter values to choose.

Example 2

The estimates of database I/O requirements for several configurations are summarized in Figure 4. The purpose of this experiment was to investigate the tradeoff between I/O time and secondary storage space for this experiment. It consisted of a single master record type (100 occurrences) and five detail record type (5000 occurrences each) with each contained in a separate chain type. The secondary

TABLE II—IDS retrieval and update application. Sample output

APPLICATION: Retrieve and Modify Record 7					
ACTION	Record Type	Chain Type	I/O Service Time (MLSEC)		System CPU Time Estimate (MLSEC)
			Single Access	Multi-Access	
Retrieve as CALC record	1	-	10.20	40.40	2.77
Retrieve as chain detail record	7	8	.07	.26	.02
Modify and delink	7	-	50.69	200.70	8.50
Retrieve master w/o currency table	2	7	10.14	40.14	2.75
Link to chain 7	7	7	20.28	80.28	2.00
Retrieve record 1 as CALC	1	-	10.20	40.40	2.77
Link to chain 8	7	8	20.28	80.28	2.00
TOTAL	-	-	121.86	482.46	20.81

storage space for this configuration was:

	CHAIN NEXT	CHAIN NEXT, PRIOR, MASTER
Master Records	100*(144 bytes)	100*(164 bytes)
Detail Records	25000*(128 bytes)	25000*(136 bytes)
Total	3214K bytes	3416K bytes

The implementation of pointers for PRIOR and MASTER in each record represents an overall increase of slightly over 6 percent in total storage space. The results illustrated in Figure 4 show that the I/O time for most operations is independent of chain pointer implementations. However, there is a slight increase in the STORE and a very significant decrease in HEAD (i.e., retrieve master) operations. The results for "multi-access" were relatively the same as those in Figure 4 but all values were approximately 30 percent larger on a point-by-point basis. The increase for STORE is due to a greater number of pointers to reset, while the decrease for HEAD is due to direct access to the master record with a MASTER pointer. Overall, this experiment shows that choice of pointer implementation depends on the types of applications to be run. Normally, a 6 percent increase in storage space is not significant, therefore, the increase in storage space appears to be less important than the effect on I/O time, which in turn affects total response time.

Example 3

We now return to the data structure for Example 1 to illustrate the capability of the DBDE to predict degradation

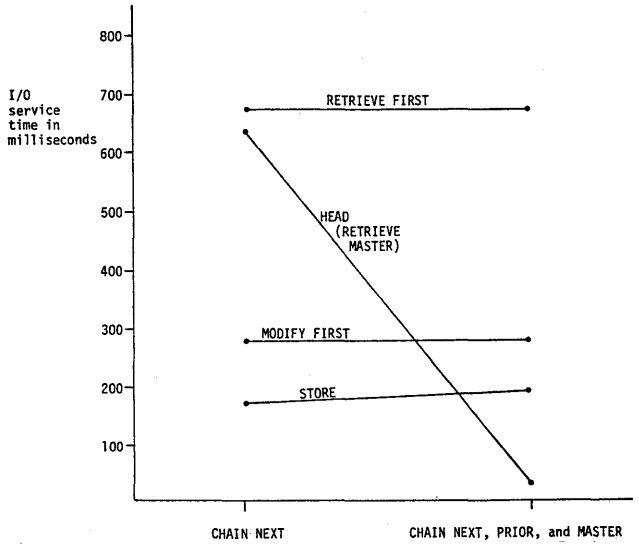


Figure 4—I/O service time for IDS operations with two pointer implementations ("single access" with dedicated disk)

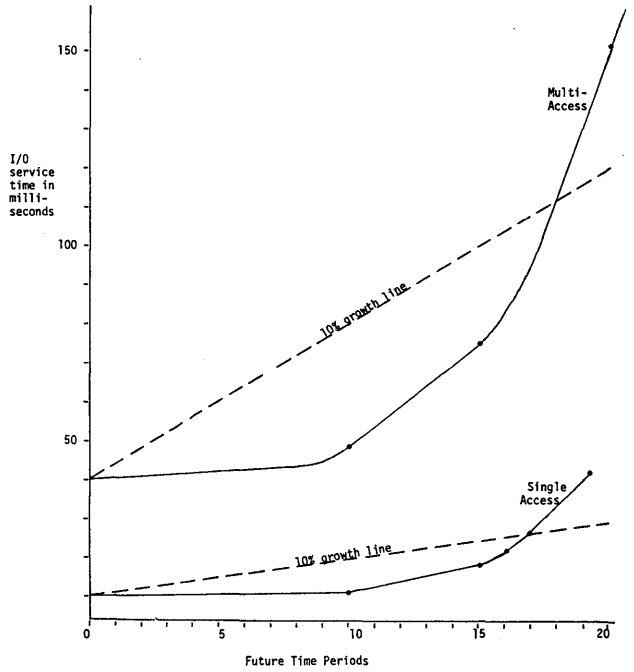


Figure 5—I/O service time for the retrieve record 7 operation as a function of increasing IDS database size

in I/O service time as the probability of overflow increases. Using the database configuration in Table 1, we assume a database growth rate (number of record occurrences of each type) of 10 percent per time period for an arbitrary time period length. The IDS model derives the overflow probability at discrete time period intervals for as many intervals as the analyst wishes to specify. The assignment of a value such as "3 days" for a time period has no effect on the model; real time units for time periods are independent of the overflow algorithm.

I/O service time to retrieve a record (type 7) is shown to degrade at a nonlinear rate over 20 time periods in Figure 5. The linear 10 percent database growth rate is included for comparison. The purpose of conducting this type of experiment is to compare storage structures, not just for the current configuration, but also for future configurations that involve more record occurrences and possibly a different level of workload. Observation of the results in Figure 5 could also help determine when and how often the database should be reorganized to clear the overflow areas.

FUTURE DIRECTIONS

Experience in the development of the DBDE has led to the conclusion that considerable insight can be gained into the causes and effects of physical database performance. Based on this experience it is felt that reasonable extensions can be made in three major directions. The first potential extension needed is to model the interface between the DBMS and the operating system in its many facets: mutual exclusion of processes attempting to read and update the

same data, queuing delays from mutual exlusion or other resource contention, and details of operating system overhead to facilitate DBMS commands to access data. The simulation model of Hulten and Soderlund[7] has achieved considerable success in this area.

Secondly, the evaluation can be extended to databases based on hierarchical or other logical models of data because they exhibit the same properties as network models: access can be described in terms of some combination of direct (hashing) or indexing plus a series of sequential or pointer steps within a particular level of indexing or at the final level of data. The generalization applies to existing search methods, indexing methods, and overflow techniques, although the resulting implementation in the analytical model is more arduous in some cases. At the level of detail exemplified by the DBDE, each technique must be modeled individually, and although models designed by Yao[14] and Severance[10] are more general, they lack important detail available in the DBDE.

The third major extension would be to incorporate the DBDE into a physical database designer that optimizes user specified performance measures as a function of user specified control parameters. Typically, the control parameters would be defined as a subset of the database characteristics currently input to the DBDE. Several of these characteristics are fixed (i.e., bound) at the logical database design phase, and so the range of physical design control parameters is further narrowed. In IDS one could control subfile page size, page range for subfiles and record types, PLACE NEAR specifications, RETRIEVAL VIA clauses, and pointer options.

ACKNOWLEDGMENTS

The research and development for the Database Design Evaluator was conducted under Defense Communications Agency Contract Number DCA 100-75-C-0064 for the WWMCCS ADP Directorate, Reston, Virginia. The authors also gratefully acknowledge the fine programming contributions made by Judy Botwick and Rick Haan.

REFERENCES

1. Bastarache, M. J. and E. A. Hershey, "The Data Base Management System User Manual and Example," *ISDOS Working Paper No. 89,* Dept. of Industrial and Operations Engin., The University of Michigan, April, 1975.
2. Cardenas, A. F., "Evaluation and Selection of File Organization—A Model and System," *Comm. ACM,* 16,9, 1973, pp. 540-548.
3. Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures," *Comm. ACM* 18,5, 1975, pp. 253-263.
4. CODASYL Data Task Group, *April 1971 Report,* ACM, New York.
5. Fry, J. P. and B. K. Kahn, "A Stepwise Approach to Database Design," *Proc. ACM Southeast Regional Conference,* Birmingham, Alabama, April 1976.
6. Honeywell Information Systems, Inc. *Integrated Data Store,* Wellesley Hills, Massachusetts, Order No. BR69, 1971.
7. Hulten, C. and Soderlund, L. "A Simulation Model for Performance Analysis of Large Shared Data Bases," *Proc. International Conf. on Very Large Data Bases,* Tokyo, Oct. 6-8, 1977.
8. Knuth, D. E. *The Art of Computer Programming, Vol. 3: Searching and Sorting,* Addison-Wesley, Reading, Massachusetts, 1973.
9. Senko, M. E., V. Lum, and P. Owens, "A File Organization Evaluation Model (FOREM)," *Proc. IFIP 1968,* pp. C19-C23.
10. Severance, D. G., *Some Generalized Modeling Structures for Use in Design of File Organizations,* Ph.D. Dissertation, University of Michigan, 1972.
11. Teorey, T. J., and K. S. Das, "Application of an Analytical Model to Evaluate Storage Structures," *Proc. ACM/SIGMOD International Conference on the Management of Data,* Washington, D.C. June 2-4, 1976, pp. 9-19.
12. Teorey, T. J., J. Botwick, R. A. Haan, and L. Oberlander, "Design Specifications for the Database Design Evaluator," Data Translation Project Working Paper DE 7.2, Graduate School of Business Admin., University of Michigan, 1977.
13. Yao, S. B. and A. G. Merten, "Selection of File Organization Using Analytic Modeling", *Proc. International Conference on Very Large Databases,* Framingham, Massachusetts, September 22-24, 1975, pp. 255-267.
14. Yao, S. B., "An Attribute Based Model for Database Cost Analysis," *ACM Trans. on Database Systems,* 2,1, March 1977, pp. 45-67.

Selection of an efficient combination of data files for a multiuser database

by R. A. DUHNE
Sistemas Computacionales Avanzados, S.H.
Monterey, Mexico

D. G. SEVERANCE
University of Minnesota
Minneapolis, Minnesota

INTRODUCTION

A database is a physical representation of information describing logical entities and relationships of interest to users. A database system functions both to *maintain* the accuracy and integrity of the representation as information changes over time, and to provide *access* to subsets of data as required by users. When choosing storage structures for a database, a designer is continually faced with tradeoffs between data maintainability and data accessibility.

The design problem is a difficult one. Typically, the number of design variations is enormous, and the choice of an efficient organization is dependent upon numerous problem characteristics:

1. the data to be stored—its sizes, volumes, and volatility;
2. the data storage environment—its operating costs and access characteristics;
3. the data retrieval applications—their frequencies, priorities, response requirements, and data selection criteria; and
4. the system's design objective—criteria for measuring performance, and an associated set of operational design constraints.

Precise mathematical formulations of realistic design problems generally reveal the presence of integer decision variables, nonlinear constraints and discontinuous objective functions. As a result, traditional mathematical optimization techniques cannot be applied to database design problems in a straightforward way.

Research aimed at systematizing the database design process is supported at a number of American universities[1,3,12,21-23] and research laboratories.[14-16] Our investigation at the University of Minnesota has evolved from research carried on first at the University of Michigan[17,18] and then at Cornell University.[7,9,11] Our approach to database design uses a model composed of (1) parametric descriptions of modular components of a generalized database organization, (2) analytic cost equations which evaluate proposed modular database designs, (3) an analyst interface which accepts and evaluates arbitrarily selected designs, and (4) search procedures which automatically generate and compare thousands of alternative organizations.

The purpose of this paper is to describe a methodology devised to permit our automatic design procedures to search a large set of alternative record access paths efficiently, while selecting the best combination for a given design problem.

THE MODELING CONTEXT

Figure 1 provides an overview of the modular, generalized database organization assumed in our research. While three major modules are distinguished, our concern here is with the design of Module A—the file structures which define access paths to subsets of database records. Results of projects to design search techniques (Module B)[4,19,20] and record structures (Module C)[2,7,10] are reported elsewhere.

We assume each retrieval request, input in Figure 1, is known and given by:

a. a relative *frequency* or priority of retrieval,
b. record *selection* criteria, based upon field values stored within data records,
c. attribute *projection* criteria defining fields to be retrieved from selected records, and
d. *ordering* criteria for the presentation of retrieved data.

To appreciate the combinatorially explosive nature of the access path selection problem, consider an employee database and the three retrieval requests described by Table I.* Any one of these requests can be satisfied by scanning all employee records in a database which is arbitrarily organized. Records of interest would be selected on the basis of DEPARTMENT and SKILL values and sorted as necessary

* Projection criteria are essential for the design of efficient record structures (Figure 1, Module C), but are ignored here.

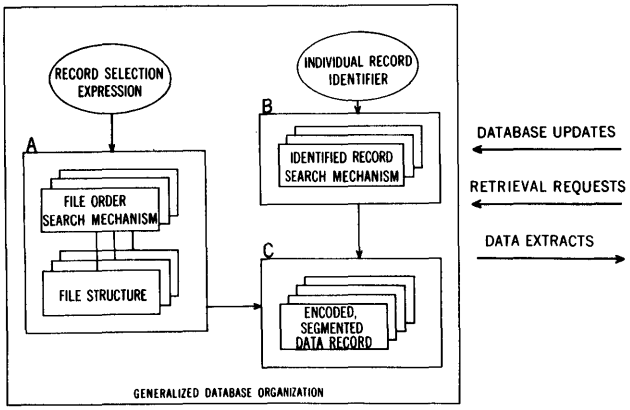


Figure 1—A component-level overview of the database design model

for presentation to the user. Since each subset is likely to be small, a number of alternative file structures might be used to organize these record collections for faster retrieval.[20] For example, one might connect the records of all engineers together in a list file in order by DEPARTMENT (Request 1), or construct an inverted list which points directly to all members of department A in order by NAME (Request 2). The "engineers in department A" (Request 3) could be retrieved via a scan and sort from either of these files, or a separate access path could be established for this request as well.

Each of the alternatives has a number of structural variations which trade record accessibility for database maintainability. Depending upon its expected rate of growth, a file may vary in its initial loading density or in its use of chained overflow areas to accommondate new records. List structures may or may not use back-pointers to facilitate record deletions. The type of pointers used within inverted files may vary depending upon the expected database volatility. Every ordered structure may use any of a number of order-value search techniques to speed insertion and deletion operations.

In general, consider a design problem with N retrieval requests, each of which on the average could be retrieved from any of K_1 distinct implementations of K_2 different file types. The problem of access path selection, then, is equivalent to choosing from among $(K_1 K_2)^N$ alternative database designs. Typical values for $K_1 K_2$ and N, in the range 20 to 50, make an exhaustive search of the solution space computationally infeasible.

TABLE I.— A Simplified Description of Three Data Extracts

Request Number	Frequency (per month)	Selection Criteria	Order Required
1	20	SKILL='ENGINEER'	DEPARTMENT
2	5	DEPARTMENT='A'	NAME
3	10	DEPARTMENT='A' and SKILL='ENGINEER'	PASS NUMBER

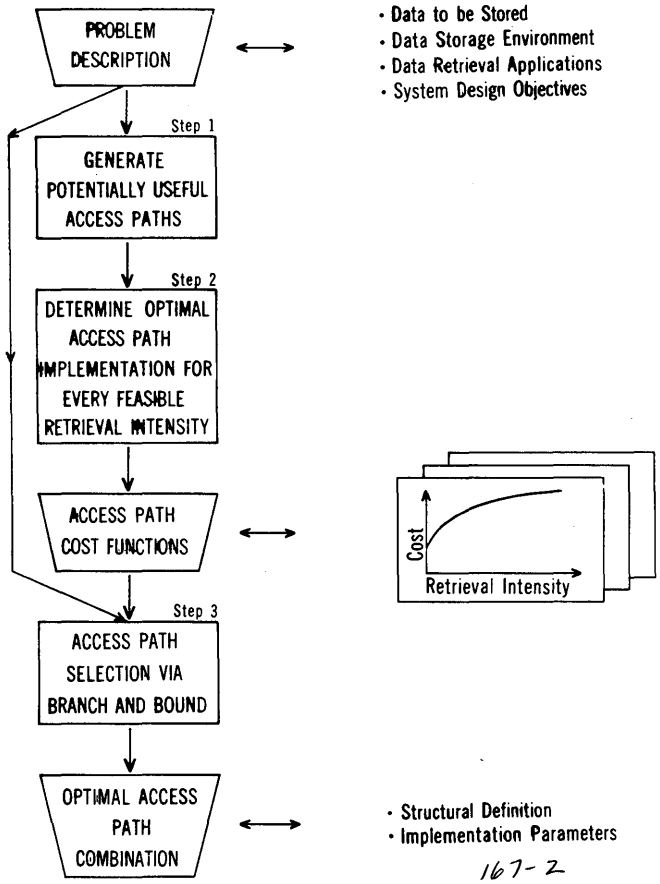


Figure 2—Overview of the access-path design procedure

THE PROCEDURE FOR DESIGNING ACCESS PATHS

A three-stage design process has been devised to select an efficient combination of access paths on the basis of the system's total cost. The procedure takes as input a quantitative description of the data to be stored, the data storage environment, and the data retrieval applications. Total cost is measured as the sum of storage, retrieval and maintenance costs. A detailed description of the analytic models used to calculate the total cost of a proposed design is well beyond the scope of this paper (Duhne[5] provides a thorough discussion). An understanding of the optimization procedure is, nevertheless, easily conveyed without substantial detail. Consider the basic design procedure as illustrated by Figure 2.

The first step in the procedure generates a large set of potentially useful access paths. Only a few will actually be selected in the final design. To insure the computational tractability of the optimization procedure, it is desirable to restrict the number of access paths considered. On the other hand, to insure the optimality of the selected design, no reasonable access path should be excluded. The set of paths is generated from the analysis of retrieval requests.[5] Each access path in the set is described generically in terms of its

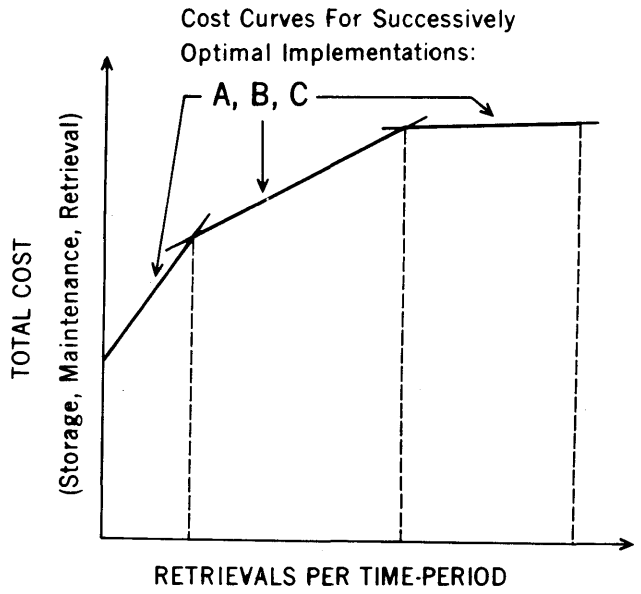


Figure 3—Typical access-path cost function

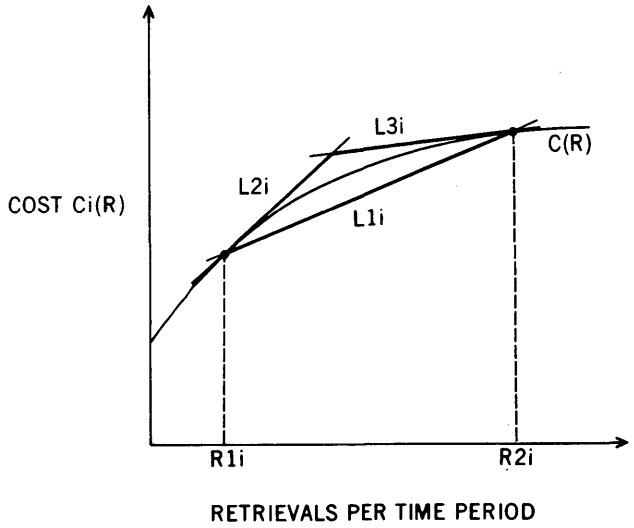


Figure 4—Cost function bounds in the interval (R1,R2)

record membership criterion, its record ordering, and its basic file type (e.g., sequential file, inverted file, etc.).

The second step shown in Figure 2 determines an optimal implementation for each potential access path at every feasible* level of retrieval activity. As was mentioned earlier, a variety of implementations trade record accessibility for data maintainability. For each level of retrieval intensity, a search procedure developed by Severance[17] and refined by Duhne[5] is used to locate an optimal implementation. Intuition correctly tells us that, as the frequency of file access increases, selected implementations will successively reduce the marginal cost for retrieving the file.

For each potentially useful access path, a cost function is output from Step 2. As illustrated by Figure 3, the function defines the minimal total cost (for storage, maintenance, and retrieval) at every level of retrieval intensity. The implementations which yield these minimal costs are stored for later recall. Duhne[5] shows that total cost is a continuous, non-decreasing, concave function of retrieval intensity. This is an intuitively reasonable property, and it is essential in the final setp of the design procedure.

The third step in Figure 2 assigns retrieval activities to access paths in a manner which minimizes total system cost. A branch-and-bound procedure is used—the reader is directed to Garfinkle and Nemhauser[8] for a thorough discussion of this optimization technique. Generally, branch-and-bound procedures implicitly enumerate all possible solutions to a problem by means of a search tree built from nodes and labeled branches. The root is associated with the complete solution space. Each other node corresponds to a restricted class of solutions derived by imposing solution restrictions which label branches on the path from the root to that node.

A search proceeds as follows: Starting with the root, a series of nodes in the tree are "visited." With each visit, upper and lower bounds on the cost of solutions reachable from the current node are computed; a least upper bound for the entire tree and its corresponding node are continually recorded. On the basis of the cost calculations at the current node, a branch is chosen for exploration. The search proceeds down the tree until a visited node is "fathomed" and no branches from that node are explored.

A node is fathomed either (1) when the upper and lower bounds for the node are sufficiently close that all solutions governed by the node are effectively equivalent, or (2) when the lower bound of the node is greater than the current least upper bound for the tree so that no solution governed by the node can be optimal.

The search algorithm "backtracks" from a fathomed node to its parent node and proceeds to search other unexplored branches in the tree. When none are left, the search is complete. The (any) solution defined by the node with the current least upper bound is selected as optimal.

This general procedure is applied to our specific problem by traversing a search tree in which each node is defined by a collection of retrieval activity intervals $\{[R1_i, R2_i]\}_{i=1}^{n}$, one for each of n potential access paths, i. Every solution governed by a node (i.e., reachable from it) must be composed of an optimal implementation for each access path given these activity intervals.* So that every possible solution is considered, the root of the search tree is defined by $\{[0, RMAX_i]\}_{i=1}^{n}$, where $RMAX_i$ is the sum of all retrieval activities which could potentially be satisfied via access path i.

An upper and lower bound on the total cost of *each* access path is computed at each node visited. Since the actual function $C_i(R)$ is known to be concave, a lower bound in the

* The maximum retrieval intensity for each access path is calculated as the cumulative frequency of all user retrieval requests which can use that path.

* If $R1_i = R2_i = 0$, then, optimally, the access path is not implemented.

interval $[R1_i, R2_i]$ (as illustrated by Figure 4) is given by the line $L1_i$ passing through $C_i(R1_i)$ and $C_i(R2_i)$. An upper bound is given by the intersecting lines $L2_i$ and $L3_i$ (passing through points $C_i(R1_i)$ and $C_i(R2_i)$ with slopes $dC_i(R1_i)/dR$ and $dC_i(R2_i)/dR$ respectively.)

Using the lower bound approximation, extracts are assigned to the access paths from which they can be retrieved at least cost. Cost is calculated as the sum† of:

(1) the cost of access path retrieval which is taken to be the slope of $L1_i$, and
(2) the cost of extract sorting, which is a function of the size and retrieval criteria of the extract, and calculated using statistics which describe the sort utility of the host system.[13]

Duhne[5] has shown that the total cost of any solution reachable from a node is bounded by the sums of upper and lower bounds for the individual access paths. If the current node is fathomed by these bounds, the search returns to its parent node. If the node is not fathomed, two new branches are generated by partitioning the retrieval interval of a selected access path into two parts. The interval to be divided is chosen so that the difference between the upper and lower bound (at the point of cumulative retrieval assignments) is greatest. The partition boundary is set at the point in this interval where the difference between its current bounds is greatest.* The search proceeds as described for the general algorithm and terminates when an optimal collection of access paths has been determined.

The design procedure has been encoded as a FORTRAN program and run on a CDC CYBER 74 at the University of Minnesota.[6] It has been applied successfully to a number of realistic design problems using less than a minute of CPU time. The cost estimates generated by our analytic models have been verified via comparisons with actual costs of design implementations. Model accuracy has ranged from one to six percent.[5]

SUMMARY

The application of mathematical optimization techniques to the difficult problem of database access path design has been described. The design process is divided into three steps. Generic descriptions of potentially useful access paths are first developed from user retrieval specifications. An optimal implementation for each access path at every feasible level of retrieval intensity is then determined. For each path a function describing minimum total cost for every retrieval intensity is constructed. Using the fact that these functions are continuous, non-decreasing, and concave, a branch-and-bound procedure is used to locate access path implementations which can be used at minimal cost for the retrieval of user extracts. The result is an efficient database design.

The design procedure has been applied to realistic problems and executes in a reasonable time. The accuracy of our analytic cost models has been verified through implementations of selected designs. Details of these models have been purposely suppressed in an effort to highlight the structure of the optimization procedure which we believe is generally applicable to the problem of access path selection. Since the procedure basically requires only that a cost function (with reasonable properties) be constructed for each access path to be considered, we feel that the algorithm can be readily adapted to the modeling work of others.

ACKNOWLEDGMENTS

This work was supported in part by the David W. Taylor Naval Ship Research and Development Center under Research Contract N00014-76-1063.

REFERENCES

1. Babad, J. M., "A Record and File Partitioning Model," *Communications of the ACM*, 20, 1, January 1977, pp. 22-31.
2. Bray, O. H., "Field Encoding Analysis Routine: User's Manual and System Documentation," *MISRC Technical Reports 77-20, 77-21*, University of Minnesota, August 1977.
3. Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures," *Communications of the ACM*, 18, 5, May 1975, pp. 253-263.
4. Carlis, J. V. and J. D. McKeen, "Search Mechanism Analysis Routine (SMAR): User's Manual and System Documentation," *MISRC Technical Reports 77-06, 77-07*, University of Minnesota, August 1977.
5. Duhne, R. A., *Optimal Design of a Generalized File Organization*, Ph.D. Dissertation, Department of Computer Science, Cornell University, 1977, 167 pages.
6. Duhne, R. A., "A Computerized File Organization Design System: System Documentation and User's Manual," *MISRC Technical Reports 77-02, 77-03*, University of Minnesota, October 1976.
7. Eisner, M. J. and D. G. Severance, "Mathematical Techniques for Efficient Record Segmentation," *Journal of the ACM*, 23, 4, October 1976, pp. 619-635.
8. Garfinkle, R. A. and G. L. Nemhauser, *Integer Programming*, New York: John Wiley and Sons, 1972.
9. Hoffer, J. A., *A Clustering Approach to the Generation of Subfiles for the Design of a Computer Data Base*, Ph.D. Dissertation, Department of Operations Research, Cornell University, 1975, 276 pages.
10. March, S. T. and D. G. Severance, "The Determination of Efficient Record Segmentations and Blocking Factors for Large Shared Databases," *ACM Transactions on Database Systems* 2, 3, September 1977.
11. Maxwell, W. L. and D. G. Severance, "Comparison of Alternatives for the Representation of Data Item Values in an Information System," *Proceedings, Wharton Conference on Research on Computers in Organizations*, University of Pennsylvania, October 1973, pp. 121-136.
12. Nunamaker, J. F., Jr., D. E. Swenson and A. B. Whinston, "Specifications for the Development of a Generalized Data Base Planning System," *AFIPS Conference Proceedings, 1973 National Computing Conference*, pp. 259-270.
13. Peeri, E., "Computer Hardware Environment Calibrator: User's Manual and System Documentation," *MISRC Technical Reports 77-18, 77-19*, University of Minnesota, 1977.
14. Schneider, L. S., "A Relational View of the Data Independent Accessing Model," *Proceedings—1976 International Conference on Management of Data*, Washington, D. C., June 1976, pp. 75-90.
15. Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structures and Accessing in Data Base Systems," *IBM Systems Journal*, 12, 1, January, 1973, pp. 30-93.

† A slightly more complex calculation can be used to model the possibility of file intersection and union during extract retrieval; see Reference 5.

* As an exception, the first partition boundary on an interval $(0, RMAX_i)$ is always set at $RMIN_i$ defined by the minimum retrieval frequency over all requests which access path i can satisfy.

16. Senko, M. E., "Data Structures and Data Accessing in Data Base Systems: Past, Present and Future," *IBM Sysytems Journal* 16, 3, September 1977, pp. 208-257.
17. Severance, D. G., *Some Generalized Modeling Structures for Use in Design of File Organizations,* Ph.D. Dissertation, University of Michigan, 1972.
18. Severance, D. G., "Identifier Search Mechanisms: A Survey and Generalized Model," *Computing Surveys,* 6, 3, September 1974.
19. Severance, D. G. and R. A. Duhne, "A Practitioner's Guide to Addressing Algorithms," *Communications of the ACM,* June 1975, pp. 314-326.
20. Severance, D. G. and J. V. Carlis, "A Practical Guide to the Selection of Record Access Paths," *ACM Computing Surveys,* December 1977.
21. Siler, K. F., "A Stochastic Evaluation Model for Database Organizations in Data Retrieval Systems," *Communications of the ACM,* 19, 2, February 1976, pp. 84-95.
22. Teorey, T. J. and K. S. Das, "Application of an Analytic Model to Evaluate Storage Structures," *Proceedings—1976 International Conference on Management of Data,* Washington, D. C., June 1976, pp. 9-20.
23. Yao, S. B., "An Attribute Based Model for Database Access Cost Analysis," *ACM Transactions on Database Systems,* 2, 1, March 1977, pp. 45-67.

Programming languages for relational data base systems*

by CHARLES J. PRENNER and LAWRENCE A. ROWE
University of California at Berkeley
Berkeley, California

INTRODUCTION

There has been a remarkable growth in the use of data base management systems over the past decade. This has resulted from the recognition that many practical applications, previously implemented by special purpose programs, can be developed more easily through the use of a general data base facility. Here, we will only be concerned with relational data base systems. The advantages of such systems have been discussed at length in the literature.[7,9,10]

In modern data base management systems, users typically interact with data bases via a query language.[4,6,8,20,41] These languages, which are often interactive, allow creation and deletion of data bases as well as retrieval from, and updating of, existing data bases. Because query languages have not provided a complete programming environment, most of these languages have been coupled to existing programming languages. Application programs consist of statements in the programming language intermixed with statements in the query language to access the data base system when required.

Previously investigated approaches to providing such access include the definition of subroutines to execute data base functions and the embedding of data base constructs into an existing language, using a preprocessor to translate these constructs into run-time calls on a data base system (e.g., EQUEL,[2] and the embedding of SEQUEL in PL/1[6]). Because the primary focus of this work was to provide data base access from a programming language, little attention was paid to the programming environment resulting from the combination of the two languages.

While a given programming language and a given query language may each be satisfactory in isolation, their combination may be less than satisfactory because neither one was designed for this more general setting. In this paper we discuss techniques for improving the programming environment for data base applications. Our hypothesis is that a superior environment can be realized by incorporating the data base operations as part of the programming language itself. In the second section we outline some limitations associated with previously proposed environments. These stem primarily from the fact that the programming language is totally unaware of the data base. In the third section we illustrate some of the benefits provided by an integrated approach through the application of current techniques from programming language research.

* This research was supported in part by U.S. Army Research Office Grant DAAG-29-76-G-0245, and by the University of California under a Faculty Research Development Grant.

CURRENT APPROACHES

Most current approaches to providing data base access from a programming language are constrained by the fact that the language system has no knowledge about the data base. We divide our discussion of the impact of this constraint into five areas: data base interface, data base operations, type system, compilation, and abstraction.

Data base interface

As mentioned previously, the two primary approaches to providing data base access have been to provide subroutines for data base functions and to embed data base constructs through the use of a preprocessor.

The subroutine approach suffers from the limitations inherent in any use of subroutines to extend the semantic space of a language; programs become essentially a series of calls which are often unreadable because of the many (bookkeeping) arguments which must be passed to the subroutine. No syntactic aids are provided to make the use of these arguments understandable.

The preprocessor approach is somewhat better in that an actual query language is used. Program statements and query language statements can be freely intermixed. In most systems the preprocessor does not have to parse the entire language because query language statements are tagged with some distinguished symbol. The preprocessor removes the query language statements, replacing them with calls upon the data base subsystem, before the program is passed to the language translator. The programmer transacts with a readable version of the program and only the language translator need be concerned with the less readable version.

Unfortunately, the preprocessor approach is limited as well. Since the query language is not part of the programming language, communication between the programming language and the data base system can become awkward.

For example, it is often necessary to use implicitly defined global variables to communicate the status of queries. In some contexts, program variables which are to be made accessible to the data base system must be preceded by "special symbols" so that they may be recognized by the preprocessor. In some preprocessors arbitrary constraints are imposed as to which arguments to a data base operation can be variables and which must be constants. In other implementations, even though the language being extended is strongly typed, the same level of checking is not extended to the data base objects and their operations. Detailed examples illustrating these problems are given in a previous paper.[35]

Although preprocessing the entire language would remove some of these limitations, this approach is still constrained by the fact that (practically speaking) no amount of preprocessing can extend the language to incorporate relations with the same amount of symmetry, type checking and protection available for built-in objects.

Data base operations

In considering the query language as part of the programming language one finds that the notation and orientation of the two are quite different. Commonly used programming languages are procedural, with built-in facilities for iteration, manipulation of complex data structures, and procedural abstraction. On the other hand, most query languages are non-procedural. In the context of a procedurally oriented programming language these non-procedural operations seem out of place. To provide a consistent programming environment the objects manipulated by the data base system and their attendant operations and control structures should be related to the objects, operations and control structures of the language itself. For example, a join operation may be thought of as a nested iteration over relations yet the operation "join" usually appears in a form completely unrelated to the iteration operations in the programming language. We do not advocate that higher level operators such as joins be excluded from a language. Instead, we suggest that they be defined as extensions to the language in terms of existing primitives.

Type system

Because the programming language has no knowledge of the data base system, the type systems of the two are usually distinct. Data base objects cannot be manipulated in the programming language with the same ease and flexibility as language objects. For example, records, as used in a relational data base system, often are not treated as composite structures in the language environment (e.g., EQUEL or SEQUEL). This means that values must be copied individually from data base records into program variables before the values can be used. Thus, a record from a data base cannot be an argument to a procedure nor an operand to an otherwise meaningful operation (e.g., record selection or the right-hand-side of an assignment). The same situation holds for relations. For example, relations may not be passed as parameters to functions. In addition, it is impossible to operate on relation types, say, to obtain the count of the number of columns in the relation, or to obtain the domain of a column as a type. The existence of two type systems may require conversions to be performed on objects as they flow between the two systems. One type system controlling data access whether in main memory or secondary storage would be more efficient because it removes the need for such conversions.

Compilation

In programming languages, there is often a tradeoff between program flexibility and run-time efficiency. Most data base systems are committed to one extreme or the other depending upon whether data base operations are interpreted (less efficient but more flexible,* e.g., INGRES) or compiled (more efficient but less flexible, e.g., SEQUEL). When a language is committed to one extreme the resulting design decisions make the language awkward to use, or less efficient, when the application requirements are not matched to that commitment. For example, consider the treatment of relation names in SEQUEL. In order for a query to appear explicitly in the text, all relation names must be constant (to allow compilation to be performed). If dynamically varying names are desired, then a string for the query text must be constructed and the relation name inserted into it at run-time. On the other hand, INGRES allows dynamically varying relation names but cannot utilize the fact that a relation name is constant in a given instance. A better programming environment would allow for a continuous spectrum ranging from dynamically varying requests with less efficient execution to fixed requests with more efficient execution so that the system can provide the best possible solution for the requirements of the problem to be solved.

Abstraction

The concept of data abstraction has received much attention in the programming language community. Recently, there has been interest in the use of data abstraction with respect to data base systems.[18] Many data base systems support abstraction related facilities such as views, integrity constraints and triggers.[3,12,19,33] For example, a view provides a form of abstraction in that the user can construct queries for relations which do not actually exist but are materialized from a number of relations. If the organization of the relations is changed, the view can be maintained by redefining it in terms of this new organization. In addition, some authors have proposed mechanisms for specifying higher level operations on data bases (such as "hire" and "fire").[18] Although some systems offer these abstraction

* Requests can be changed at run-time (e.g., in EQUEL relation and domain names can change dynamically).

facilities, they have not yet integrated them with their counterparts in data abstraction from programming languages.*

AN INTEGRATED APPROACH

We will now consider the integration of data base objects and operations into a programming language. We will not attempt to give a complete language proposal here. Instead, we shall illustrate some of the benefits of this approach. Our discussion is divided into four areas: type system, data base operations, compilation, and abstraction.

Type system

An extremely important aspect of the language is the extension of the type system to provide access to data base relations residing on secondary storage. While a complete discussion of the type system is beyond the scope of this paper, we will discuss a few aspects of the type system to demonstrate the usefulness of an integrated approach. We have been strongly influenced in our thinking about types by the Mesa language.[15]

In introducing data types for relations in the programming language we will exploit the obvious correspondence between relations and collections of records in programming languages, namely, a relation may be viewed as an unordered collection of records (tuples in data base terminology), where the collection may not contain duplicate tuples. For example,

```
Employee: type=relation of
                    name: string(20),
                    dept: integer,
                    salary: integer,
                    manager: string(20),
                    jobtitle: string(15)
                  end
```

defines Employee as a relation type, where *"relation of"* means "unordered collection with no duplicates of . . .".

A relation type is itself an object upon which operations may be performed to obtain access to types contained within it. Thus,

```
declare e: record(Employee)
```

declares e as a variable whose type is identical to the underlying record type of the relation, and

```
declare deptno: Employee.dept
```

declares deptno to be a variable whose type is identical to the dept column of the relation, i.e. integer.

By allowing the types contained within the relation type to be extracted in this manner, duplication of these types in the program is avoided. Thus, a degree of modularity is achieved. In addition, data integrity is enhanced because the only operations that can be performed on the values in the data base are those allowed on the given type. Because there is only one representation for a value of a given type, whether in the data base or in the program environment, and because the set of types provided in the language is the same as those provided in the data base system, no conversion is needed when data is moved from one environment to the other.

Another important feature of the type system is the ability to specify constant or dynamically varying relation and domain names. For example, the following program fragment declares Rel to be a relation bound to the existing data base relation EMP.

```
begin
  declare Rel: Employee=relation('EMP');
    ⋮
end;
```

If the relation were not constant, e.g., if its name were entered at a terminal, the program fragment would be

```
begin
  declare Rel: Employee,
          Rname: string(15);
  write("What is the relation name?")
  read(Rname);
  Rel: =relation(Rname)
    ⋮
end;
```

where *relation* is a function which takes a string argument and returns a relation as result (or returns *undefined* if no such relation exists). In either case the type of the relation is specified so that operations on Rel can be type-checked, and the program code in the block can be the same. The only change is in the way the relation is bound to Rel. Because the programmer has explicit control over the conversion of the input string into an object of type relation, the existence of the named relation can be tested and an appropriate action taken if it does not exist. This cannot be done easily in any existing language.

The last aspect of the type system to be discussed is type extensibility. Several researchers have mentioned that a type extension facility would be valuable so that types other than the typical primitives (e.g., integers, reals, and character strings) can be placed into a data base record. The proposed language will allow arbitrary fixed length structured objects, except those including pointers or relations, to be a component of a data base record. Providing this feature is not particularly difficult if the data base system supports a typing mechanism which can be extended to include generic functions.*[39]

* Recent work in this direction has been reported.[28,32]

* Generic functions have multiple definitions for an operator where the particular definition invoked at a call is determined by the arguments (e.g., + has several definitions, including definitions for integer, real, complex or mixed arguments).

Data base operations

There are several ways that data base operations can be made more convenient, consistent, and more closely related to existing constructs in programming languages. Our interest in this research was motivated by the difficulty one author of this paper had expressing complicated queries in SEQUEL. This led to the design of a procedural query language, described in detail elsewhere.[26] In the following discussion, an overview of the query constructs is presented.

Suppose that we wish to print the names of all employees whose manager is Kathy Fallon. This is written as

```
for x in Rel st x.manager='Kathy Fallon' do write(x.name);
end
```

and read "for each record x in the relation Rel, such that x's manager is 'Kathy Fallon', print x's name."* The variable x is known only in the scope of this statement and is bound to each record in the relation satisfying the qualification. The qualification clause (*st*) is optional.

We believe that this construct is cleaner than those used in many existing data base languages. The binding of language environment variables to data base objects is more consistent and the control structure is more natural, say, than requiring explicit testing of a variable shared by the data base system and the program environment to determine if another record exists.[3]

A query which requires the join of two relations may be written as a nested *for* construct. For example, a list of all employees who earn more than their managers could be obtained by writing

```
for emp in Rel do
  for mgr in Rel st emp.manager=mgr.name do
    if emp.sal>mgr.sal then write(emp.name) fi
  end
end
```

Suppose one wanted to construct a temporary relation having employee records for all people working in department 50. This could be accomplished by using the *for* construct to append qualifying tuples to a (temporary) relation variable.

Alternatively, the *for* statement can be used in its short form

```
begin
  declare Reltemp:employee;
  RelTemp:=[all | Rel:dept=50]
    ⋮
end
```

which says "construct a relation composed of those records in Rel such that dept is 50 and assign it to RelTemp." The construct on the right hand side of the assignment operation is an example of a *relation constructor*. The keyword *all* indicates that all fields of the record are selected. A relation constructor causes a copy of each record to be made. If only the name and salary fields were needed, the expression would be

```
[name,salary | Rel:dept=50]
```

Of course, to assign this value to RelTemp, the type specification for RelTemp must be changed.

The value returned by a relation constructor may contain duplicate records. Such objects are called bags,[27] or multisets.[23] If the value is assigned to a relation variable, then duplicates are removed. However, there are situations in which the value is useful with duplicates retained, e.g., when a function such as average is used as in AVG([salary|rel]). To explicitly remove duplicates from the value, two vertical bars ("||") are used. A relation constructor may be used directly in a *for* statement without having been previously bound to a relation variable. This is demonstrated by the next example which intermixes query iteration, relation constructors, and functions which take relations as arguments to print a list of all departments and the average salary of employees in that department.

```
for x in [dept || Rel] do
  write(x.dept,AVG([salary|Rel:dept=x.dept]));
end
```

This is read "for x in the set of distinct departments from Rel, print x and the average salary for those employees in department x".

The iterative construct is also used to update individual records as shown in the next example in which all employees are given a 10 percent raise.

```
for x in Rel do
  x.salary:=x.salary * 1.1;
end
```

As in most relational systems, the relation is not actually changed *until all iterations are completed* because the storage structure for the relation may require that records be physically moved when a field is changed, in which case a record could be updated more than once. One solution, used in some set theoretic languages, is to make a temporary copy of the set over which the iteration variable ranges.[13] This would not be practical in this environment because of the size of relations and the time needed to make a copy. Alternatively, the problem may be solved by keeping all changes in a separate file until the end of all iterations.[34] This also solves the problem of keeping a data base consistent after certain kinds of crashes.

Compilation

The linguistic benefits provided by an integrated environment will be of little utility in practice unless reasonably efficient code can be produced. Here, we discuss a number of compilation techniques which can be utilized in order to ensure that programs in the language have an efficient realization. The topics discussed are: variable binding time,

* The use of a *for* statement to sequence through a collection of data objects has been proposed before by programming language designers[13,17,21,30,31] but is absent from existing data base languages.

partial evaluation of procedures at compile-time, and program optimization.

Variable binding time

Binding time is the time at which a variable is bound to an entity, e.g., a specific relation to a variable of type relation. Whether this binding occurs at compile-time or run-time influences both run-time efficiency and program flexibility. Figure 1 shows a graph of binding time of variables in data base queries versus the degree to which these queries are compiled for two relational data base systems.

Compiling is more efficient in execution time and may be more efficient in execution space, while late binding allows more flexibility. Notice that these two systems make explicit commitments, to flexibility (INGRES) or to execution efficiency (SEQUEL). By using compilation and optimization techniques developed for implementing programming languages, it is possible to develop a system that operates in the grey region of the graph. When enough parts of a language construct are fixed (e.g., the relation name is constant) code as efficient as possible can be compiled.[22] Otherwise, less efficient code is compiled. This means that the tradeoff between efficiency and flexibility can be controlled by the applications programmer. Furthermore, it means that only those programs that need more flexibility must pay for it in terms of efficiency. Of course, in those cases where code is compiled to access a specific relation, programs may have to be recompiled if the definition of the relation changes.

Partial evaluation

A second compilation technique which can be used is the partial evaluation of procedures at compile-time (procedure closure). In SEQUEL, all relation names are viewed as either constants in the program text (in which case compilation is performed) or as strings "read in" at run-time (in which case no compilation is performed). With this organization it is impossible to pass a relation name to a procedure and to compile queries issued from the procedures which use this relation name. However, with global flow analysis techniques[16] combined with procedure closure techniques,[37] a programmer may be given a number of choices as to how such a procedure call should be compiled. These include:

1. compile one procedure body using only the type information about the relation,
2. compile distinct procedure bodies for each of the different (constant) relations passed to the procedure using type information as well as specific information about the relation, or
3. expand certain calls "in-line" so that no call is actually made.

Providing a programmer with a facility such as this gives one considerable control over the tradeoff between execution time and space.

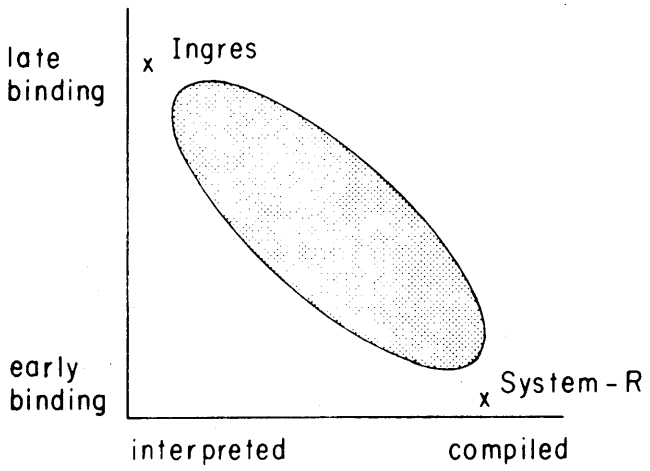


Figure 1—Binding time versus compilation

Optimization

The use of an integrated language opens up many possibilities for optimization. The goal is to reduce the number of calls on the data base system (and subsequent disk accesses) by combining many queries into one, removing queries from loops, eliminating temporary relations, and so on. This can be achieved through the application of traditional programming language optimization techniques (e.g., code motion, common subexpression elimination, and loop fusion).[1] However, the potential payoff from such optimizations is much greater in data base applications than in typical programs because the removal of a disk access represents a savings many orders of magnitude greater than the removal of, for example, an assignment statement.

Beyond traditional types of optimizations, possibilities exist for more complex optimizations using techniques similar to those used in experimental programming systems which have program verification capabilities.[38] In this case, semantic information available in an abstraction containing relations (e.g., integrity constraints) and information about the physical organization of the data can be used to perform sophisticated optimizations.

Abstraction

Data abstraction facilities have been found useful in developing programs that involve complex data structures.[11,24,40] It seems reasonable to consider extending such facilities to handle relational objects and operations. Another reason for exploring such a facility is because it may provide a structuring mechanism for current data base concepts such as views, integrity constraints, and triggers[3,12,19,33] which are all related to abstraction. Currently these facilities appear in data base systems in an unstructured fashion which often makes it difficult to determine how they interact and relate to one another. A better approach would be to make them all aspects of a single abstraction mechanism.

A view permits a programmer to conceive of a data base in terms of virtual relations. This allows fields, or complete relations, to be operated on even though they might not exist physically. This is similar to the notion of uniform reference that is being explored in programming languages.[14] Another use of views is to prevent a programmer from seeing certain columns of a relation. Again this is similar to a programming language concept (see the *private* attribute in Mesa[15]). A problem with using views is that it is not always clear how a value to be stored to a virtual field should be decomposed and stored into the fields that are physically present. Another problem is that it is not always clear what action should be taken if a value is to be assigned to a field which would cause the record to fall out of the view. Some data base researchers propose to solve these problems by establishing defaults where unambiguous alternatives are possible and disallowing the operations otherwise. An alternative solution, which we are currently investigating, is to acknowledge the difficulty of establishing defaults and instead provide convenient tools for allowing the implementer of the abstraction to specify the semantics explicitly.

Triggers are data base operations which are invoked automatically when certain primitive operations are performed, e.g. when a tuple is deleted from the employee relation the count of employees in the removed employee's department is decremented by one. In the context of a data abstraction facility it would seem more reasonable to associate this operation with the abstract operation "fire" than with the deletion operation itself.

Integrity constraints are assertions which characterize what it means for the data base to be in a correct and consistent state with respect to the semantic meaning of the data. For example, a constraint might be that all the names in the data base must be a subset of those in the employee relation.[19] Currently, such constraint information is used in an enforcement sense, i.e., a constraint is checked after a data base operation (or a group of operations) are performed. Often these checks are very expensive. It may be possible to use this information as assertions to be proved to hold about the set of abstract operations which are permitted on the data base. For those constraints which cannot be proved to always hold it may be possible to automatically determine additional information which should be retained in order to simplify enforcement.

CONCLUSIONS

We have attempted to show that an integrated approach to providing data base access from a programming language can yield a better programming environment for data base applications than previously available. We are currently in the process of designing a language with this philosophy for use with the INGRES[20] system. Other research groups are also working on similar problems.[5,25,29,36] We believe that this research represents a significant step towards a better understanding of the relationship between programming languages and data base systems.

REFERENCES

1. Aho, A. V. and J. D. Ullman, *Principles of Compiler Design,* Addison-Wesley, Reading, MA, 1977.
2. Allman, E., M. R. Stonebraker and G. Held, "Embedding a Relational Data Sublanguage in a General Purpose Programming Language," Proceedings of a Conference on Data: Abstraction, Definition, and Structure, *SIGPLAN Notices,* 11, Special Issue, March 1976, pp. 25-35.
3. Astrahan, M. M., et al., "System R: Relational Approach to Data Base Management," *ACM Transactions on Database Systems,* 1, 2, June 1976, pp. 97-137.
4. Boyce, R. F., D. D. Chamberlin, W. F. King and M. M. Hammer, "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage," *Communications of the ACM,* 18, 11, November 1975, pp. 621-628.
5. Bratbergsengen K. and O. Risnes, "ASTRA—A DBMS Based on a High Level, Relational DML with Data Access via a Hierarchical DDL," *Proceedings SIMULA Users Conference,* September 1977.
6. Chamberlin, D. D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal of Research and Development,* 20, 6, November 1976, pp. 560-575.
7. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* 13, 6, June 1970, pp. 377-387.
8. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings 1971 ACM-SIGFIDET Workshop,* San Diego, CA, November 1971, pp. 35-68.
9. Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers," *Proceedings 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* Ann Arbor, Michigan, May 1974.
10. Date, C. J. and E. F. Codd, "The Relational and Network Approach: Comparison of the Application Programming Interfaces," *Proceedings of the 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control.* Ann Arbor, Michigan, May 1974.
11. Dahl, O.-J. and C. A. R. Hoare, "Hierarchical Program Structures," in *Structured Programming,* Academic Press, New York, NY, 1972.
12. Eswaren, K. P., "Aspects of a Trigger Subsystem in an Integrated Database System," *Proceedings Second International Conference on Software Engineering,* San Francisco, CA, October 1976, pp. 243-250.
13. Feldman, J., J. Low, D. Swinehart and R. Taylor. "Recent Developments in SAIL—An Algol Based Language for Artificial Intelligence," *Proceedings 1972 Fall Joint Computer Conference,* Vol. 41, Las Vegas, NV, December 1972, pp. 1193-1202.
14. Geschke, C. M. and J. Mitchell, "On the Problem of Uniform References to Data Structures," *IEEE Transactions on Software Engineering,* SE-1, 2, June 1975, pp. 207-219.
15. Geschke, C. M., J. H. Morris, Jr. and E. H. Satterthwaite, "Early Experience with Mesa," *Communications of the ACM,* 20, 8, August 1977, pp. 540-552.
16. Graham, S. L. and M. Wegman, "A Fast and Usually Linear Algorithm for Global Flow Analysis," *Journal of the ACM,* 23, 1, January 1976, pp. 172-202.
17. Gries, D. and N. Gehani, "Some Ideas on Data Types in High-Level Languages," *Communications of the ACM,* 20, 6, June 1977, pp. 414-420.
18. Hammer, M., "Data Abstractions for Data Bases," Proceedings of a Conference on Data: Abstraction, Definition, and Structure. *SIGPLAN Notices,* 11, Special Issue, March 1976, pp. 25-35.
19. Hammer, M. and D. McLeod, "A Framework for Data Base Semantic Integrity," *Proceedings Second International Conference on Software Engineering,* San Francisco, CA, October 1976, pp. 498-504.
20. Held, G., M. R. Stronebraker, and E. Wong, "INGRES—A Relational Data Base System," *Proceedings 1975 National Computer Conference,* 44, Anaheim, CA, May 1975, pp. 409-416.
21. Hoare, C. A. R., "Notes on Data Structuring," in *Structured Programming,* Academic Press, New York, NY, 1972.
22. Katz, R. H., "Compilation of Data Base Queries," M.S. Project, Department of Electrical Engineering and Computer Sciences, U.C. Berkeley, June 1978.
23. Knuth, D. E., *The Art of Computer Programming, Volume 3.* Addison-Wesley, Reading, Mass, 1973.

24. Liskov, B., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," *Communications of the ACM*, 20, 8, August 1977, pp. 564-576.
25. Merrett, T. H., "Aldat—Augmenting the Relational Algebra for Programmers," Technical Report SOCS-78.1, School of Computer Science, McGill University, November 1977.
26. Prenner, C. J., "A Uniform Notation for Expressing Queries," ERL Memorandum M77/60, Electronics Research Laboratory, U.C. Berkeley, September, 1977.
27. Rulifson, J. F., et al., "QA4: A Language for Writing Problem Solving Programs," *Proceedings IFIP Congress*, 1968.
28. Schmidt, J. W., "Type Concepts for Database Definition: An Investigation Based on Extensions to Pascal," Institut fur Informatik, Universitat Hamburg, June 1977.
29. Schmidt, J. W., "Some High-level Language Constructs for Data of Type Relation," *ACM Transactions on Database Systems*, 2, 3, September 1977, pp. 247-261.
30. Schwartz, J. T., "On Programming: An Interim Report on the SETL Project. Installment 1—Generalities. Installment 2—The SETL Language and Examples of its Use," Computer Science Department, Courant Institute of Mathematical Sciences, New York University, 1973.
31. Shaw, M., W. A. Wulf, and R. L. London, "Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators," *Communications of the ACM*, 20, 8, August 1977, pp. 553-563.
32. Smith, J. M. and D. C. P. Smith, "Integrated Specifications for Abstract Systems," Technical Report UUCS-77-112, Computer Science Department, University of Utah, September 1977.
33. Stonebraker, M. R., "Implementation of Integrity Constraints and Views by Query Modification," *Proceedings 1975 ACM-SIGMOD Workshop on the Management of Data*, San Jose, CA, May 1975.
34. Stonebraker, M. R., E. Wong, P. Kreps and G. Held, "The Design and Implementation of INGRES," *ACM Transactions on Database Systems*, 1, 3, September 1976, pp. 189-222.
35. Stonebraker, M. R. and L. A. Rowe, "Observations on Data Manipulation Languages and their Embedding in General Purpose Programming Languages," *Proceedings Third International Conference on Very Large Data Bases*, Tokyo, Japan, October 1977.
36. Wasserman, A. I. and E. Handa, "Preliminary Report on the Programming Language PLAIN," Medical Information Science, U.C. San Francisco, January 1978.
37. Wegbreit, B., "Procedure Closure in EL1," Center for Research in Computing Technology, Harvard University, March 1972.
38. Wegbreit, B., "Multiple Evaluators in an Extensible Programming System," *Proceedings 1972 Fall Joint Computer Conference*, 41, Anaheim, CA, December 1972, pp. 905-915.
39. Wegbreit, B., "The Treatment of Data Types in EL1," *Communications of the ACM*, 17, 5, May 1974, pp. 251-264.
40. Wulf, W. A., R. L. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Transactions on Software Engineering*, SE-2, 4, December 1976, pp. 253-265.
41. Zloof, M. M., "Query-by-Example," *Proceedings 1975 National Computer Conference*, 44, Anaheim, CA, May 1975, pp. 431-438.

Conversion of high-level sublanguage queries to account for database changes*

by STANLEY Y. W. SU and MICHAEL J. REYNOLDS
University of Florida
Gainesville, Florida

INTRODUCTION

Over the past few years, technological advances in database management have introduced many new research and development problems. The problem of "application migration" of a database management system is one of them. It can be grossly divided into two closely related subproblems: the problem of database translation and the problem of application program conversion due to database changes. There are many reasons for database reorganization and restructuring. Some of these are (1) the change of the use environment of a DBMS, (2) the upgrading of hardware and software facilities, (3) the change from one database system to another, (4) the installation of a new DBMS, (5) the modification of an existing system for efficiency purposes, and (6) the change of database semantics. The database translation problem has been tackled with some success by the works reported in References 7-9, 13-16, 18, 20-24, 29.

Some database changes, both structural and semantic ones, may require that the set of application programs written for the old database be modified or converted to account for the changes either for semantic reasons or for efficiency purposes. A well-designed DBMS, which is data independent, should normally not be affected by the changes made on the physical structure and access strategy because these types of database changes can be handled simply by changing the mappings from the external model to the conceptual model and from the conceptual model to the internal model.[6] However, few existing database systems are truly data independent. Physical placement of data and access strategies are often carried into the application programs.[3,4,12] Furthermore, some semantic changes of database will inevitably cause changes in both the schema and the subschema which the data sublanguage directly interacts.

Very often, the sublanguage statements as well as the host language statements need to be modified. As with the area of database translation, application program conversion could feasibly be implemented by the user or system programmer, however, the task is such an enormous one that the personnel's time is not effectively optimized. Therefore, a need for some type of program conversion system is definitely warranted.

* This research is supported by NSF Grant MCS 76-10075.

The work on application program conversion is still in its infancy. In the paper by Su,[25] a brief review of some existing works related to this problem was given. An attempt was made to analyze the problem and to identify the tasks which need to be undertaken to obtain some kind of automatic or semi-automatic aid for program conversion. In the same work, a general model of program conversion as it is related to database translation was presented. It was proposed that two stages of conversion are involved with the program conversion problem in the DBMS environment:

(1) Sublanguage query conversion whereby query analysis and modification occur due to schema and subschema changes, and
(2) Host language program conversion whereby the tracing of program logic, the identification of variable relations and program-subprogram structure represent some of the needed analyses.

Since the problem of program conversion is such a complicated one, we have decided to narrow our initial investigation to a restricted area in the first stage of program conversion, i.e., the sublanguage query conversion. In order to get a better handle on the problem, the following assumptions and restrictions are made: (1) The data model used in the DBMS is the relational model.[5] (2) The subschema is identical to the schema of the database. (This assumption does not nullify the problem investigated here because of the types of database changes considered here.) (3) The DEFINE language[10] is used as the data description language. (4) The CONVERT language[19] is used as the data mapping language. The CONVERT language is designed for processing and converting hierarchical trees called "forms." It is borrowed here for relational processing because relations are special cases of general "forms." (5) The relational sublanguage SEQUEL[1,2] is used to formulate the source and target queries. Only the basic patterns of the language are considered.

The objectives of this investigation are as follows: (1) To study the effects of all the CONVERT operators, (2) to design the algorithms for collecting data from the input source to construct the data tables to form the translation dictionary, (3) to study the techniques for analyzing a limited yet general SEQUEL query patterns, and (4) to design a

translation algorithm in detail (i.e., to the level of algorithm description which is one step short of the actual coding) to support the analysis and translation of queries.

This paper is a short version of a technical report printed in May, 1977.[26] Related to this work, two independent works on program conversion (Su and Liu[27] and Housel[11]) take a more general approach to the program conversion problem. Both works advocate the use of an abstract representation of program semantics and the semantics of data translation operators.

In the subsequent section, we shall first describe several types of database changes and illustrate the problems of sublanguage query conversion with examples which involve the use of CONVERT operators and SEQUEL's basic mappings. A translation process is then detailed in the third section and an example of query conversion is provided in the fourth section to clarify and illustrate the translation process step-by-step. A conclusion and the future work are given in the last section.

SOME ORGANIZATIONAL AND SEMANTIC CHANGES OF A DATABASE

We shall describe in this section some organizational and semantic changes and illustrate how these changes can be brought about by the CONVERT operators and their effects on the SEQUEL queries. This description and illustration should give the reader a feel of the sublanguage conversion problem we are dealing with and of the CONVERT and SEQUEL languages used in this study. The data changes described here constitute only a small subset of possible changes and the description of the languages is not complete. The interested reader should refer to the original papers on the language.[1,2,19]

In the following presentation we shall use the following relations for illustration purposes.

EMP (E#, ENAME, SEX, MGR)
PROJECT (P#, PTITLE, BUDGET, DEPARTMENT)
E-P (E#, P#)

A. A Large Relation is Split into Several Relations

This type of organizational change may occur in a DBMS for data access efficiency and data security reasons. For example, the PROJECT relation may be split into several small relations, one for each of the departments handling the projects. This may need to be done because it was observed that most of the application programs access departmental projects separately and the use of different security constraints are required for different departments' projects. To perform such a database translation, the following CONVERT statements can be used. Here, we assume that there are three departments only.

```
PROJ-D1←SELECT (P#, PTITLE, BUDGET FROM
        PROJECT:
        PROJECT.DEPARTMENT=ACCOUNT-
        ING)
PROJ-D2←SELECT (P#, PTITLE, BUDGET FROM
        PROJECT:
        PROJECT.DEPARTMENT=SALES)
PROJ-D3←SELECT (P#, PTITLE, BUDGET FROM
        PROJECT:
        PROJECT.DEPARTMENT=PURCHASING)
```

Each SELECT statement selects the tuples satisfying the DEPARTMENT qualification and creates a new relation with a different name. Note that the DEPARTMENT information now becomes implicitly specified by the new relation names.

The effects of this type of database changes are many. A SEQUEL query for accessing the tuples of the original PROJECT relation such as

```
SELECT    *
FROM      PROJECT
WHERE     BUDGET>50K
```

would have to be converted into the following structure.

```
SELECT    *
FROM      PROJ-D1
WHERE     BUDGET>50K
U
SELECT    *
FROM      PROJ-D2
WHERE     BUDGET>50K
U
SELECT    *
FROM      PROJ-D3
WHERE     BUDGET>50K
```

The retrieved result is still different from the original query because the department information is not explicitly specified in the new database. The DEPARTMENT field and value need to be added into the records retrieved by each SEQUEL basic mapping (the SELECT-FROM-WHERE block). This can only be done if the conversion system stored the DEPARTMENT information of PROJ-D1, PROJ-D2 and PROJ-D3 files during the analysis of the CONVERT operators. Queries making use of the DEPARTMENT information or retrieving the information will have to be specially handled.

B. Several Relations are Combined into a Single Relation

This type of database reorganization is necessary in order to eliminate the unnecessary duplications and to facilitate the access of related data scattered in several relations. This is often required when changing a conventional file system into a database system.

To reverse the operation given in (A), the following CONVERT statements can be used.

```
CONCAT (DEPARTMENT: ACCOUNTING ONTO
  PROJ-D1 AT BUDGET)
CONCAT (DEPARTMENT: SALES ONTO PROJ-D2
  AT BUDGET)
CONCAT (DEPARTMENT: PURCHASING ONTO
  PROJ-D3 AT BUDGET)
PROJECT←MERGE (PROJ-D1, PROJ-D2, PROJ-D3)
```

The CONCAT operator expands the relations by inserting the department data to the right of the BUDGET field. It makes the DEPARTMENT data explicit in each project relation. Here, the arguments, DEPARTMENT: ACCOUNTING, DEPARTMENT: SALES and DEPARTMENT: PURCHASING, are used in place of relation names which are required in the original CONVERT syntax. The effect of this type of change to the SEQUEL queries is that references made to several relations should be changed so that reference can now be made to the same relation. This generally causes a compression of the number of basic mappings to a single mapping such as the reverse operation of the example shown in (A).

C. Changes of the mathematical mappings between/among Entities

A database can be said to contain data about some entities and their associations. Several mathematical mappings between/among entities have been identified: (1) one-to-one association between two entities or within one entity, (2) one-to-many association between two entities or within one entity, (3) many-to-many associations between two entities or within one entity and (4) many-to-many associations among more than two entities. Each type of association can be changed to any other types due to rule or policy change of an enterprise using the database. For example, the rule which allows many employees to participate in a project and an employee to participate in many projects (many-to-many associations between EMP and PROJECT entities) can be changed so that one employee can only participate in one project but one project can be participated in by several employees (thus, a one-to-many association between PROJECT and EMP entities). To represent the new semantics, the relations EMP (E#, ENAME, SEX, MGR, P#) and PROJECT (P#, PTITLE, BUDGET, DEPARTMENT) would be sufficient. If the database contents have been changed by data deletion so that one employee participates in only one project and it was further assumed that the data of employees not participating in any project are to be deleted, the CONVERT operator, GRAFT, can be used for this database change. GRAFT provides a means of combining two or more relations into one when specific conditions are satisfied. For example, the following statement

```
NEW←GRAFT (E-P ONTO EMP: E-P. E# =
  EMP.E#)
```

would create a new relation, NEW (E#, ENAME, SEX, MGR, P#), which contains the data of each employee plus the project he participates in. If this relation is used to replace the EMP and E-P relations in the target database, the program translator needs to know at least three things: (1) the data of those employees who have not participated in the projects have been deleted from the database, (2) the EMP relation has been expanded to include P# data and has been renamed, and (3) the many-to-many association has been changed to a one-to-many association.

A SEQUEL query which retrieves the project numbers of all projects participated by SMITH from the source database will have to be modified as follows:

```
SELECT  P#                          SELECT  P#
FROM    E-P              ⟶         FROM    NEW
WHERE   E# IN                       WHERE   ENAME='SMITH'
        SELECT  E#
        FROM    EMP
        WHERE   ENAME='SMITH'
```

The nested mappings on the left are compressed and modified to account for the database changes as shown in the new query on the right. In this example, the translation analyst will have to be informed of the possible loss of data due to the deletion caused by the database change. If the facts of data deletion have been recorded, the program conversion system can tell the analyst precisely what data are missing by running the new query against the new database.

D. Adding or Deleting Entities and/or Associations

This type of database change requires the modification of schema and subschema and often requires changes to the application programs. It reflects the dynamic nature of a database to meet the new requirements of migrating applications. Adding new data generally will not affect the application program in a well-designed DBMS. Nevertheless, it can affect the storage operations (delete, insert and update) of some existing systems. For example, in DBTG's network model, if a new record type is added into a database as the AUTOMATIC member of an owner-coupled set whose owner was in the database, a deletion statement involving the owner in an old application program may have to be modified so that the new record occurrences will not be mistakenly deleted because of their AUTOMATIC membership specification. Deleting data (entity and association data) from database will always entail changes to the sublanguage statements as well as the host language program itself. Any reference to the deleted data will have to be taken care of by the conversion system. The translation analyst needs to be informed of the deletion and references to the deleted data can be taken out of the program if agreeable to the analyst. However, this kind of program modification may cause a chain of reactions to other program statements since they may make use of the deleted data or those data which depend on the deleted data. Furthermore, some deletions may render the sublanguage statements meaningless. For example, a set of sublanguage statements

may originally be designed to process data about two entities through their association and the association of these two entities might have been deleted. The statements would no longer be meaningful. In this situation, the statements should be taken out and the translation analyst informed of the program changes made.

The foregoing discussion represents a sample of a large number of situations in which changes in the subschema and thus the application programs are either required or desirable. It also illustrates the types of changes needed to be made in the sublanguage queries due to database changes. In the examples given, we consider only the effects of a single database conversion operator in isolation. In reality, the problem is much more complex than what has been presented since a set of CONVERT operators might have been applied to the source database. In this situation, the program conversion system has to keep track of all the possible effects on each domain and each relation of the source database. It also has to keep track of the conditions under which each relation in the target database is created and how it is related to the source relations. Sizable data need to be collected from the description of source and target databases and from the database conversion operators applied.

A SEMI-AUTOMATIC ANALYSIS PROCESS FOR SUBLANGUAGE QUERY CONVERSION

The entire analysis process is divided into six stages as illustrated in Figure 1.

A. *Stage 1—Translation Dictionary Development*

The contents of the translation dictionary represent information extracted from the source and target database descriptions and from the data mapping operators original applied to generate the target database. The DEFINE data descriptions and the CONVERT statements represent the two sources from which the information in the dictionary is constructed. The information is useful in the conversion of all source database queries. It guides the formal translation algorithm (Stage 5) in the construction of new queries which will account for the database changes. The translation dictionary contains seven tables.

(1) TABLE NAME REASSIGNMENTS (TNR): This table is used primarily to determine if any source database table names have been changed during the data conversion process. It has the following format: TNR(SOURCE_TABLE_NAME, TARGET_TABLE_NAME). Changes in database table names occur mainly in the CONVERT language ASSIGNMENT statement, however implied changes are also possible in just about all of the CONVERT language statements as shown in the following examples. The source tables renamed are underlined.

```
TABLE0 ← EMP(EMP#,DEPT,STATUS)
SLSPR2(DEPT#,TOTSALES) ← SALES(DEPT,
TOTSLS)
TABLE1←SELECT(DEPT,NAME FROM DPTTBL:
        DPTTBL.D# = SUP.D#)
TABLE2←SLICE(E#,EMPNAME,SALARY FROM
        PERSONNEL)
TABLE3←GRAFT(INVNTY ONTO PTS:
        PTS.P# = INVNTY.P#)
TABLE4←CONCAT(TBL10,TBL20 ONTO TBL30 AT
        S#)
TABLE5←MERGE(SERVICE1,SERVICE2)
TABLE6←SORT(TSBLONE BY S#,P#)
TABLE7←ELIM-DUP(TABLETWO)
TABLE8←CASE(TABLE15.SEX='Male','Female')
        (SELECT(E#,'1', FROM TABLE15)
         SELECT(E#,'0', FROM TABLE15))
```

The contents of this table is needed mostly in the reassigning process of the FROM clause of a SEQUEL language basic mapping. The SEQUEL language FROM clause refers to the database table from which the query in question will derive its results.

(2) DOMAIN NAME REASSIGNMENT (DNR): This table records the changes on the source domain names. The data is used primarily during Stage 4 to develop the query pretranslation reference tables which will be described later. The table has the following format: DNR (SOURCE_DOMAIN_NAME, TARGET_DOMAIN_NAME). Changes in the database domain names occur mainly in the CONVERT language ASSIGNMENT statement, however implied changes are also possible in just about all of the CONVERT language statements. Some examples are given below. The respective source domain names are underlined.

```
DATABL(STAFF#,PART) ←——— DATATLB(EMP#,ITEM)
EXMPTBL ←——— TABLE20(S# RENAME TO SUPPLIER, P# RENAME TO PART
TABLE25 (DPMT,IDNET) ←——— SELECT(DEPT,NAME FROM DEPTTBL)
TABLE26 (BOOK#,CATLG) ←——— SLICE(SERLNM, INDEX FROM LIBRYTB)
TABLE27 (EMP#,SEX) ←——— CASE (EMPTBL.SEX='MALE', 'FEMALE')
                         (SELECT(E#, '1', FROM EMPTBL),
                          SELECT(E#, '0', FROM EMPTBL))
```

The contents of this table is needed mostly in the reassigning process of the SELECT, WHERE, AND, and OR CLAUSES of a SEQUEL language basic mapping. All of these clauses of the SEQUEL basic mapping contain database domain name references which relate to the database table present in the FROM CLAUSE of the very same basic mapping.

(3) TABLE DOMAIN USAGE (TDU): This dictionary

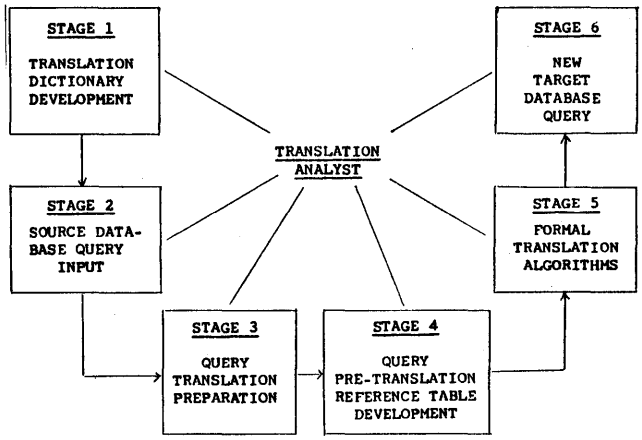


Figure 1—Stages of query analysis and conversion

table has the dual purpose of specifying all of the domains which a database table contains and all of the database tables which contain a certain domain (Figure 2).

This example shows database table TABLE1 as containing domains EMP#, MGR and SALARY; PERSONNEL as containing domains EMP# and MGR: and, PRODUCT as containing domains EMP# and SALARY. This dictionary table helps in the query conversion process by making sure that all references to database tables and domains in the target database are legitimate. It also helps in the reassigning of domain names and table names of SEQUEL basic mapping which have had extensive name reference changes.

(4) TABLE CONVERSION INDICATORS (TCI): This table specifies the CONVERT operators which have been applied on the relations during data conversion. The data is useful for determining the order of execution of the formal algorithms in stage 5 of the analysis process. The structure of this table is illustrated in Figure 3.

(5) CASE ASSIGNMENT DEFINITIONS (CAD): The CONVERT CASE ASSIGNMENT statement allows for varied operations to occur over different instances of a database table. In essence, it may be desirable to code a value or values of a given domain of a database table in a different manner for usage in the target database. Therefore, these new values need to be stored in the CAD table so that when the source domain values are referred during the query conversion process, the new values may be inserted in their place. For example, the following CONVERT statement recodes the sex field of the Personnel table (male=1, female=0) and changes the domain name from SEX to SEX-CODE and the table name PERSONNEL to TABLE 1,

```
TABLE    1    (EMP#,AGE,SEX-CASE(PERSONNEL.SEX='MALE','FEMALE')
                CODE,RACE)←(SELECT(EMP#,AGE,'1',RACE FROM PERSONNEL),
                            SELECT(EMP#,AGE,'0',RACE FROM PERSONNEL))
```

The above statement causes two entries in the CAD table shown in Figure 4.

The CAD table is accessed by the formal query conversion algorithms only after all source database table and domain names have been reassigned to their corresponding target database references.

(6) GRAFT CONVERT INFORMATION (GCI): This table is related to the CONVERT GRAFT operation. Entries in the GCI table simply show which database table was grafted onto which table during data conversion. The information present in this table contributes greatly to the restructuring process of source database queries. In general, source queries using database tables which have been grafted together may be able to be restructured into a much simpler query.

(7) SELECT OCCURRENCE CONDITIONS (SOC): This table holds information concerning the conversion of source tables which were operated upon by the SELECT operator. For example, the following two SELECT operations will produce the entries shown in Figure 5.

TDU

	TABLE1	PERSONNEL	PRODUCT
EMP#	1	1	1
MGR	1	1	0
SALARY	1	0	1

Figure 2—Table for domain usage

TCI

	ASSIGNMENT	SLICE	SELECT	GRAFT	...	CASE ASSIGNMENT
EMP	1	0	1	0		0
SALES	0	1	0	1		0
TABLE1	0	0	0	0		1

Figure 3—Table for conversion indicators

CAD

TARGET TABLE NAME	TARGET DOMAIN NAME	SOURCE DOMAIN VALUE	TARGET DOMAIN VALUE
TABLE1	SEX-CODE	MALE	1
TABLE1	SEX-CODE	FEMALE	0

Figure 4—Table for case assignment

```
SELECT(S#,DES,SNAME FROM
  SLSPR:SLSPR.S#>10)
SELECT(P#,PNAME,SUPL FROM PTS:PTS.P#=50)
```

The information in the SOC table is used to determine if a query which makes reference to relations that have been operated upon by the SELECT operator will produce any result.

B. *Stage 2—Source Database Query Input*

This stage deals with the analysis of the source query to determine if the query is one of the appropriate structures capable of being operated upon by the formal conversion algorithm; and, to verify the labeling of the SEQUEL basic mappings present within the source query since the elementary unit of conversion by the formal translation algorithm is the basic mapping. Because of the degree of structural flexibility within the SEQUEL language and the original concern to narrow down the problem to a manageable size, it was decided to curtail the number of acceptable query structures. The acceptable SEQUEL structural patterns are shown below. It is felt that the structures selected are generalized enough to allow for the processing of a wide range of source queries.

ACCEPTABLE SEQUEL QUERY PATTERNS

TYPE I: BASIC MAPPING WITH NO POSSIBLE NESTING REFERENCES:

```
SELECT   "Domain Name(s)"
FROM     "Table Name"
WHERE    "Conditional Phrase"
AND      "Conditional Phrase"
OR       "Conditional Phrase"
```

TYPE II: BASIC MAPPING WITH POSSIBLE OUTER AND/OR INNER REFERENCE MAPPINGS:

```
[POSSIBLE OUTER
 MAPPING—
 TYPE II STRUCTURE]=
                    SELECT   "Domain Name(s)"
                    FROM     "Table Name"
                    WHERE    [POSSIBLE INNER MAPPING—
                    AND       TYPE II STRUCTURE OR
                    OR        CONDITIONAL PHRASE]
```

TYPE III: BASIC MAPPING WITH CONNECTING INTERSECTION AND/OR UNION OPERATORS:

```
SELECT   "Domain Name(s)"
FROM     "Table Name"
WHERE    [POSSIBLE INNER MAPPING—
AND       TYPE II STRUCTURE OR
OR        CONDITIONAL PHRASE]
∪,∩
SELECT   "Domain Name(s)"
FROM     "Table Name"
WHERE    [POSSIBLE INNER MAPPING—
AND       TYPE II STRUCTURE OR
OR        CONDITIONAL PHRASE]
. . . . . further segments are possible
```

NOTE: in all of the structural types presented above, the AND and OR clauses are optional.

For ease of description and because the formal translation algorithms need to be specific in addressing the basic parts of a query, it is necessary to develop some terminology associated with the contents and structure of a SEQUEL query. We will refer to a basic SEQUEL mapping as a "segment." Therefore, depending upon the nesting factor of a query structure, a particular basic mapping can be defined within a query by its *level* and *segment* numbers. There can be as many levels as appropriately needed for an application. The level within a query changes only when an outer or inner mapping occurs in the query itself. Different basic mappings used in one level are noted as different segments. For example, the query presented below would be labeled as follows:

```
[LEVEL 1          SELECT    SALES,TAX
 SEGMENT 1] ____  FROM      TABLEONE
                  WHERE     DEPT=
[LEVEL 2                    SELECT    DEPT
 SEGMENT 1]_________________FROM      STORETB1
                            WHERE     MGR='ALAN'
                            ∪
[LEVEL 2                    SELECT    DEPT
 SEGMENT 2]_________________FROM      STORETB2
                            WHERE     LOCTYPE=2
                  AND       SALESMAN=
[LEVEL 2                    SELECT    SALESMAN
 SEGMENT 3]_________________FROM      TABLETWO
                            WHERE     SALARY≥11000
```

The information content of each segment (basic mapping) within each and every level of a query is coded and saved so that mapping relationships and information may be available for the formal algorithm. The following information is derived for every segment within a given level (Figure 6): In order to label the information provided above in its level and segment context, the following syntax will be used:

[ABBREVIATED REFERENCE NAME, L,S]

Now, for a full understanding of the meaning of the query labeling terminology, an example will be presented. Note that since some of the label references provided are optional in the SEQUEL query syntax, the value in such a case where there is an applicable information will be 'null.' Say the query in question was the following:

```
SELECT   EMP#,CODE
FROM     TABLEONE
WHERE    DEPT=5
AND      SALES+COMM<1000
AND      MGR='ALLEN'
∪
SELECT   EMP#
FROM     TABLETWO
WHERE    E#=
         SELECT    E#
         FROM      TABLETHREE
         WHERE     SUM(SALES)=10,000
         AND       DEPT+CODE=5
         OR        SALARY
                     +COUNT(T#)=20
         OR        MARK=1
```

SOC

SOURCE TABLE NAME	SOURCE DOMAIN NAME	SPECIAL CONDITION	
		VALUE	OPERATOR
SLSPR	S#	10	>
PTS	P#	50	=

Figure 5—Table for selection conditions

INFORMATION DESCRIPTION	ABBREVIATED REFERENCE NAME
LEVEL NUMBER	L
SEGMENT NUMBER	S
OUTER MAPPING REFERENCING CLAUSE DOMAIN	OMRCD
SELECT CLAUSE DOMAIN (S)	SELDOM(N)
FROM CLAUSE TABLE	FROMTBL
WHERE CLAUSE DOMAIN (S)	WHRDOM(N)
WHERE CLAUSE ARGUMENT	WHRARG
AND CLAUSE (S) DOMAIN (S)	ANDDOM(C,N)
AND CLAUSE (S) ARGUMENT	ANDARG(C)
OR CLAUSE (S) DOMAIN (S)	ORDOM(C,N)
OR CLAUSE (S) ARGUMENT	ORARG(C)
INTERSECTION-UNION CONNECTION	INTUNC

Figure 6—Information contents for each segment

Then the following labels represent some examples of information which could be derived from the example query:

```
[SELDOM(1),1,1]    = 'EMP#'
[SELDOM(2),1,1]    = 'CODE'
[FROMTBL,1,1]      = 'TABLEONE'
[WHRDOM(1),1,1]    = 'DEPT'
[WHRARG,1,1]       = '5'
[ANDDOM(1,1),1,1]  = 'SALES'
[ANDDOM(1,2),1,1]  = 'COMM'
[ANDDOM(2,1),1,1]  = 'MGR'
[ANDARG(2),1,1]    = 'ALLEN'
[OMRCD,1,1]        = 'null'
[INTUNC,1,1]       = 'U'
[SELDOM(1),1,2]    = 'EMP#'
[FROMTBL,1,2]      = 'TABLETWO'
[OMRCD,1,1]        = 'E#'
[WHRDOM(1),1,2]    = 'E#'
[FROMTBL,2,1]      = 'TABLETHREE'
[WHRDOM(1),2,1]    = 'SALES'
[ORDOM(1,2),2,1]   = 'T#'
[ORARG(2),2,1]     = '1'
```

C. *Stage 3—Query Translation Preparation*

Stage 3 of analysis determines which translation procedures need to be used in converting each and every basic mapping within the query. There are four separate procedures designed to carry out the translation task. The order and number of procedures to be used in converting a basic mapping depends totally upon the source database table being referred to in the FROM clause. Using the TCI table, it is possible to search for the source database table to determine which CONVERT operations were performed. The selection of procedures and their order of execution depend on the CONVERT operators applied on the source database table as illustrated by the flowchart in Figure 7. Thus, if TABLE 1 of a segment were operated upon by the CONVERT operators SORT and CASE ASSIGNMENT, PROCEDURE1 and PROCEDURE3 would be called to translate the segment.

D. *Stage 4—Query Pre-Translation Reference Table Development*

During this stage, a set of reference tables is set up based on the analysis of a source database query. For each basic mapping in a query, the following number of tables are established. Their contents are described below.

REFTB1

This reference table contains information about the domain(s) used in the WHERE clause of a basic mapping. Each tuple of REFTB1 contains a WHERE clause domain (or corresponding domain name reassignment) and the target database table in which the domain can be found. This information can be extracted from the TDU Dictionary Table.

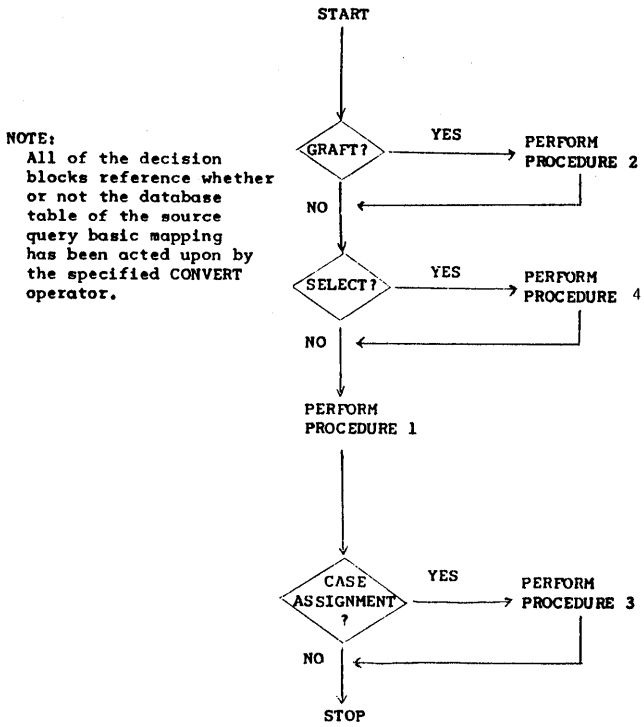


Figure 7—Program selection and execution sequence

REFTB2.n

This is a set of reference tables (n=0,1,2, . . .) each of which is set up for an AND clause of the basic mapping. The information stored in these reference tables concerns the domain or domains used in the AND clauses and the target database tables in which they are contained.

REFTB3

The contents of this reference table are derived from an analysis of REFTB1 and REFTB2.n tables. It is primarily used to determine if there is any target database table which contains all of the domains present within the WHERE and AND clauses of the basic mapping.

REFTB4.n

This set of tables (n=0,1, . . .) has the similar contents as the REFTB2.n except that the tables relate to the domain(s) in the OR clauses of the basic mapping.

REFTB5

The contents of this reference table are derived from an analysis of the target database tables which are present in REFTB3 and REFTB.n. It is used to determine if there is any target database which contains all of the domains present within the WHERE, AND and OR clauses of the basic mapping.

REFTB6 THROUGH REFTB19

These reference tables hold information concerning the domain(s) of the SELECT clause of the basic mapping being analyzed. It is assumed that a maximum of 14 domains can appear in a SELECT clause. Each table contains the name of a domain, all its domain reassignment names and the target database tables in which they can be found.

REFTB20

The contents of this table are generated by an analysis performed over REFTB6 through REFTB19. It lists the names of all target database tables in which all the domains used in the SELECT clause can be found.

Q.C.T.

This is a special reference table whose initials represent "Query Conversion Table". It is used directly by the translation algorithm to restructure the source database query. Its contents are developed through an analysis of REFTB5 and REFTB20. Query changes due to domain and table name reassignments are noted in this table.

The above set of tables are different from the Dictionary Tables generated in Stage 1. The dictionary tables hold information concerning the entire source database and the data conversion information; whereas, the reference tables contain data related to one source query. Examples of the reference tables are given in the next section where the entire translation process is illustrated by an example.

E. *Stage 5—Formal Translation Algorithm*

Stage 5 uses the data established in the dictionary and reference tables to convert each of the basic mappings in the source query. The basic mappings are processed sequentially from top to bottom (i.e., the segment with the lower level number first). As mentioned earlier, there are four main procedures used for query translation. These procedures have been written in PL/1-like statements which describe the algorithms of conversion and are given in a master's thesis by Reynolds.[17] PROCEDURE2 is designed for taking care of the effect of applying the GRAFT operator. PROCEDURE 3 and 4 are concerned with CASE ASSIGNMENT and SELECT operations respectively. PROCEDURE1 is a general procedure which handles the conversion of basic SEQUEL mappings to account for database changes introduced by other CONVERT operators.

A sizable number of subroutines and functions have been defined for accessing data from the dictionary and reference tables, for testing the data conditions specified in the basic mappings of the source query and for generating the basic mappings of the target query. These routines and functions are described in the original report.[26] They are not given here because of their length.

F. *Stage 6—New Target Database Query*

This stage is concerned with the formation of the new target database query. The labeling techniques and inner workings of the translation procedures make the final construction of the new target query rather simple to derive. There are only two distinct ways in which a query's input structure may be modified: expanding a basic mapping into two mappings and contracting two basic mappings into one. These two basic structural changes are handled by RESTRICT3 and RESTRICT5 routines.

RESTRICT3 is concerned with query transformations caused by changes in data access paths. When a query structure conversion of this type occurs, the nesting factor or total number of levels present within the query increases. While this expansion is taking place, the function also readjusts the labeling criteria of all basic mappings within the source database query to compensate for the transformation. All reference tables are also reassigned so that they are associated with the correct basic mappings. There is no need to develop reference table information for the newly created basic mapping levels. RESTRICT3 ensures the correctness of these particular basic mappings.

RESTRICT5 is concerned with query transformations caused by the use of the GRAFT operator on database tables during the data conversion process. In a structural conversion of this kind, the nesting factor or total number of levels present within a query decreases. RESTRICT5, like RESTRICT3 also readjusts the labeling criteria of all basic mappings within the source database query to account for this structural contraction. All reference tables are either reassigned or redeveloped so that proper association is recovered.

Thus, with all structural conversions being taken care of by RESTRICT3 and RESTRICT5, the same basic mapping processing sequence developed during Stage 2, modified by any added and/or deleted basic mappings, can be used to construct the new target query. This stage is performed after all basic mappings of a source query have been processed by Stage 5.

AN EXAMPLE ON THE SUBLANGUAGE QUERY CONVERSION

A query conversion example presented within this section is given in order that the reader may obtain a clearer understanding of the analysis process and the translation algorithm. More examples of different degrees of complexity can be found in a Master's thesis by Reynolds[17] where the degree of complexity of the examples is varied depending upon which CONVERT operators were used on the database tables referenced by the queries and the query structure itself. For our example, we shall first present, by illustration, the source database contents, the CONVERT operations used on the source database and the newly formed target database contents. The query will then be put through the analysis process and the results of the stage-by-stage analysis will be given. The Stage 5 results are only summarized. They show the outcome of tracing the procedure steps and functions.

Source Database

EMPLINFO

EMP#	DEPT	PHONE	EXPER	AGE	SEX	RACE
1	TOYS	25791	3	20	M	W
2	MENS	28192	3	45	F	N
3	LADIES	29999	1	36	M	A
4	SHOES	28181	1	29	M	W

MGRINFO

EMP#	DEPT	PHONE	EXPER	AGE	SEX	RACE
100	MENS	33456	6	40	M	A
200	TOYS	37878	6	38	M	N
300	LADIES	35512	6	62	M	W
400	SHOES	35418	7	56	M	W

EMPSIDAL

EMP#	NAME	SALARY	COMM	DEPEND
1	TERRY KERRIGAN	12000	4	1
2	NITA CALLAHAN	13000	3	1
3	JEFF ICARDI	08000	1	2
4	TIM MALONE	07000	1	2

MGRIDSAL

EMP#	NAME	SALARY	BENEFITS	DEPEND
100	CHRIS CLEARY	15000	1	0
200	JOHN HAUSER	15000	1	0
300	SCOTT RAYBURN	15000	1	1
400	PAUL GIOIA	18000	2	8

DEPTSALES

DEPT	SALES	INVENTY
MENS	36009	1024
TOYS	10112	5890
LADIES	29001	6950
SHOES	12152	0500

MONTHBILLS

COMPANY	BILL	USAGE	CODE
KENNEYS	04560	00010	A
SCHAFFERS	01130	00195	B
LEVIS	00050	00080	A
KTEL	10905	01059	C

SERVICES

ITEM	COMPANY	ADDRESS	PHONE	BILLDATE
01	KENNEYS	GAINESVILLE	3781125	01
02	SCHAFFERS	MIAMI	4542121	08
03	LEVIS	ATLANTA	3771156	15
04	KTEL	MIAMI	5518907	25

SPECITEM1

ITEM	DEPT	CODE1	PRICE	QTY
01	MENS	001	00215	30
02	TOYS	001	00389	20
03	LADIES	033	00111	40
04	SPORTS	033	04450	15
05	AUTO	010	00889	67

SPECITEM2

CODE2	ADVERTZ	SUPPLIER
AA	1	101
BA	1	111
BB	2	222
CD	4	321

Source Database Convert Operations

(1) EMPIDENT←EMPIDSAL(EMP#,NAME)
(2) MGRIDENT(MGR#,NAME)←MGRIDSAL(EMP#,NAME)
(3) EMPIDSAL(EMP#,SALARY,COMM)
(4) MGRIDSAL(MGR#,SALARY,BENEFITS)←MGRIDSAL(EMP#,SALARY,BENEFITS)
(5) SELECT(ITEM,DEPT,CODE1,PRICE,QTY FROM SPECITEM1:CODE1='001' OR CODE1='033')
(6) GRAFT(MONTHBILLS ONTO SERVICES: MONTHBILLS.COMPANY=SERVICES.COMPANY)
(7) SERVICE1(COMPANY,AMTDUE,USAGE,CODE,ITEM,LOCATION,PHONE,BILLDATE)←SERVICES
(COMPANY,BILL,USAGE,CODE,ITEM,ADDRESS,PHONE,BILLDATE)
(8) CONCAT(SPECITEM2 ONTO SPECITEM1 AT QTY)
(9) STAFFINFO←MERGE(EMPLINFO,MGRINFO)
(10) SORT(DEPTSALES BY INVENTY)
(11) STAFFINFO←CASE(STAFFINFO.SEX='M','F')
(SELECT(EMP#,DEPT,PHONE,EXPER,AGE,'MALE',RACE FROM STAFFINFO),
SELECT(EMP#,DEPT,PHONE,EXPER,AGE,'FEMALE',RACE FROM STAFFINFO))

Target Database

EMPIDENT

EMP#	NAME
1	TERRY KERRIGAN
2	NITA CALLAHAN
3	JEFF ICARDI
4	TIM MALONE

MGRIDENT

MGR#	NAME
100	CHRIS CLEARY
200	JOHN HAUSER
300	SCOTT RAYBURN
400	PAUL GIOIA

EMPIDSAL

EMP#	SALARY	COMM
1	12000	4
2	13000	3
3	08000	1
4	07000	1

MGRIDSAL

MGR#	SALARY	BENEFITS
100	15000	1
200	15000	1
300	15000	1
400	18000	2

SPECITEM1

ITEM	DEPT	CODE1	PRICE	QTY	CODE2	ADVERTZ	SUPPLIER
01	MENS	001	00215	30	AA	1	101
02	TOYS	001	00389	20	BA	1	111
03	LADIES	033	00111	40	BB	2	222
04	SPORTS	033	00450	15	CD	4	321

SERVICE1

COMPANY	AMTDUE	USAGE	CODE	ITEM	LOCATION	PHONE	BILLDATE
KENNEYS	04560	0010	A	01	MIAMI	3781125	01
SCHAFFERS	01130	0195	B	02	ATLANTA	4542121	08
LEVIS	00050	0080	A	03	GAINESVILLE	3771156	15
KTEL	10905	1059	C	04	MIAMI	5518907	25

STAFFINFO

EMP#	DEPT	PHONE	EXPER	AGE	SEX	RACE
1	TOYS	25791	3	20	MALE	W
2	MENS	28192	3	45	FEMALE	N
3	LADIES	29999	1	36	MALE	A
4	SHOES	28181	1	29	MALE	W
100	MENS	33456	6	40	MALE	A
200	TOYS	37878	6	38	MALE	N
300	LADIES	35512	6	62	MALE	W
400	SHOES	35418	7	56	MALE	W

DEPTSALES

DEPT	SALES	INVENTY
SHOES	12152	0500
MENS	36009	1024
TOYS	10112	5890
LADIES	29001	6950

STAGE 1: Translation Dictionary Development

TNR

SOURCE TABLE NAME	TARGET TABLE NAME
EMPIDSAL	EMPIDENT
MGRIDSAL	MGRIDENT
MONTHBILLS	SERVICE1
SERVICES	SERVICE1
SPECITEM2	SPECITEM1
EMPINFO	STAFFINFO
MGRINFO	STAFFINFO

DNR

SOURCE DOMAIN NAME	TARGET DOMAIN NAME
EMP#	MGR#
BILL	AMTDUE
ADDRESS	LOCATION

TDU

	EMP IDENT	MGR IDENT	EMP IDSAL	MGR IDSAL	SPEC ITEM1	SER VICE1	STAFF INFO	DEPT SALES
EMP#	1	0	1	0	0	0	1	0
NAME	1	1	0	0	0	0	0	0
MGR#	0	1	0	1	0	0	0	0
SALARY	0	0	1	1	0	0	0	0
COMM	0	0	1	0	0	0	0	0
BENEFITS	0	0	0	1	0	0	0	0
ITEM	0	0	0	0	1	1	0	0
DEPT	0	0	0	0	1	0	1	1
PRICE	0	0	0	0	1	0	0	0
QTY	0	0	0	0	1	0	0	0
CODE2	0	0	0	0	1	0	0	0
ADVERTZ	0	0	0	0	1	0	0	0
SUPPLIER	0	0	0	0	1	0	0	0
COMPANY	0	0	0	0	0	1	0	0
AMTDUE	0	0	0	0	0	1	0	0
USAGE	0	0	0	0	0	1	0	0
CODE	0	0	0	0	0	1	0	0
LOCATION	0	0	0	0	0	1	0	0
PHONE	0	0	0	0	0	1	1	0
BILLDATE	0	0	0	0	0	1	0	0
EXPER	0	0	0	0	0	0	1	0
AGE	0	0	0	0	0	0	1	0
SEX	0	0	0	0	0	0	1	0
RACE	0	0	0	0	0	0	1	0
SALES	0	0	0	0	0	0	0	1
INVENTY	0	0	0	0	0	0	0	1
CODE1	0	0	0	0	1	0	0	0

TCI

	ASSIGN	C/E	SELECT	SLICE	GRAFT	CONCAT	MERGE	SORT	ELIM DUP	CON SOLI DATE	CASE ASSIGN MENT
EMPLINFO	1	0	0	0	0	0	1	0	0	0	1
MGRINFO	1	0	0	0	0	0	1	0	0	0	1
EMPIDSAL	1	1	0	0	0	0	0	0	0	0	0
MGRIDSAL	1	1	0	0	0	0	0	0	0	0	0
DEPTSALES	0	0	0	0	0	0	0	1	0	0	0
MONTHBILLS	0	0	0	0	1	0	0	0	0	0	0
SERVICES	1	1	0	0	1	0	0	0	0	0	0
SPECITEM1	0	0	1	0	0	1	0	0	0	0	0
SPECITEM2	0	0	0	0	0	1	0	0	0	0	0

CAD

TARGET TABLE NAME	TARGET DOMAIN NAME	SOURCE DOMAIN VALUE	TARGET DOMAIN VALUE
STAFFINFO	SEX	M	MALE
STAFFINFO	SEX	F	FEMALE

GCI

SOURCE TABLE NAME	ONTO TABLE NAME
MONTHBILLS	SERVICES
SERVICES	SERVICES

SOC

SOURCE TABLE NAME	SOURCE DOMAIN NAME	SPECIAL VALUE	CONDITION OPERATOR
SPECITEM1	CODE1	001	=
SPECITEM1	CODE1	033	=

STAGE 2: Source Database Query Input*

```
L1,S1--------SELECT   NAME
             FROM     EMPIDSAL
             WHERE    EMP# =
L2,S1 -------------------SELECT     EMP#
                        FROM       EMPLINFO
                        WHERE      EXPER > 1
                        AND        AGE ≤ 36
                        OR         RACE = 'A'
             AND        SALARY  ≥  9000
```

* This query is recognized as one of the possible structures capable of being operated upon by the formal conversion algorithms.

STAGE 3: Query Translation Preparation

(1) The database table of the LEVEL 1, SEGMENT 1, basic mapping was acted upon by both the ASSIGNMENT and COMPONENT EXTRACTION CONVERT operations. Thus perform:
PROCEDURE 1

(2) The database table of the LEVEL 2, SEGMENT 1 basic mapping was acted upon by ASSIGNMENT, MERGE and CASE ASSIGNMENT. Thus perform the following algorithms in the order specified:
PROCEDURE 1
PROCEDURE 3

STAGE 4: Query Pre-Translation Reference Table Development

LEVEL 1, SEGMENT 1:

REFTB1

WHERE CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
EMP#	EMPIDSAL
EMP#	EMPIDENT
EMP#	DEPTSALES
MGR#	MGRIDENT
MGR#	MGRIDSAL

REFTB2.0

AND CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
SALARY	EMPIDSAL
SALARY	MGRIDSAL

REFTB3

WHERE CLAUSE DOMAIN(S)	REFTB2.0 AND CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
EMP#	SALARY	EMPIDSAL
MGR#	SALARY	MGRIDSAL

REFTB4.0

OR CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
	empty

REFTB6

SELECT CLAUSE DOMAIN	TARGET DATABASE TABLE
NAME	EMPIDENT
NAME	MGRIDENT

LEVEL 2, SEGMENT 1:

REFTB1

WHERE CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
EXPER	STAFFINFO

REFTB2.0

AND CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
AGE	STAFFINFO

REFTB3

WHERE CLAUSE DOMAIN(S)	REFTB2.0 AND CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
EXPER	AGE	STAFFINFO

REFTB4.0

OR CLAUSE DOMAIN(S)	TARGET DATABASE TABLE
RACE	STAFFINFO

REFTB6

SELECT CLAUSE DOMAIN	TARGET DATABASE TABLE
EMP#	EMPIDENT
EMP#	EMPIDSAL
EMP#	STAFFINFO
MGR#	MGRIDENT
MGR#	MGRIDSAL

REFTB5

WHERE CLAUSE DOMAIN(S)	REFTB2.0 AND CLAUSE DOMAIN	REFTB4.0 OR CLAUSE DOMAIN	TARGET DATABASE TABLE
EXPER	AGE	RACE	STAFFINFO

STAGE 5: Formal Translation Algorithm

Query Basic Mapping Processing Order:

L1,S1
L2,S1

*BEGIN PROCESSING OF L1,S1 BASIC MAPPING:

```
SELECT  NAME
FROM    EMPIDSAL
WHERE   EMP# =
AND     SALARY ≥ 9000
```

PROCEDURE 1:

	TEST	*RESULT*	*ACTION*
STEP 1:	INRMP(L,S,WHERE)	TRUE	NONE
	REFTBL2(L,S,1)	FALSE	PROCEED
STEP 2:	CLAUSE(L,S AND)	TRUE	N=2.0 J=1
	INRMP(L,S,AND)	FALSE	NONE
	REFTBL2(L,S,2.0)	FALSE	PROCEED
STEP 3:	CLAUSE(L,S,OR)	FALSE	NONE
STEP 4:	OUTMP(L,S)	FALSE	NONE
	DOMVAR(L,S,SELECT)	FALSE	NONE
	COMNTAB(L,S,6,19)	TRUE	SET REFTB20 = REFTB6
STEP 5:	REFTBL2(L,S,1) AND REFTBL3(L,S,2.0,2.9)	FALSE	NONE
	REFTBL2(L,S,3)	FALSE	PROCEED
STEP 6:	REFTBL2(L,S,5)	TRUE	NONE
	REFTBL3(L,S,4.0,4.9)	TRUE	SET REFTB5 = REFTB3
STEP 7:	COMNTAB2(L,S,5,20)	FALSE	NONE
	OUTMP(L,S)	FALSE	NONE
	REFTBL2(L,S,5)	FALSE	NONE
	REFTBL2(L,S,20)	FALSE	K=1, M=1 RESTRCT3(L,S)
	TNR2(EMPIDSAL,EMPIDENT)	TRUE	PROCEED

L1,S1 PROCESSING RESULTS:

```
SELECT  NAME
FROM    EMPIDENT
WHERE   EMP# =
        SELECT      EMP#
        FROM        EMPIDSAL
        WHERE       EMP# =
        AND         SALARY ≥ 9000
```

*BEGIN PROCESSING OF L2,S1 BASIC MAPPING:

```
SELECT  EMP#
FROM    EMPLINFO
WHERE   EXPER > 1
AND     AGE ≤ 36
OR      RACE = 'A'
```

PROCEDURE 1:

	TEST	*RESULT*	*ACTION*
STEP 1:	INRMP(L,S,WHERE)	FALSE	NONE
	REFTBL2(L,S,1)	FALSE	PROCEED
STEP 2:	CLAUSE(L,S,AND)	TRUE	N=2.0 J=1
	INRMP(L,S,AND)	FALSE	NONE
	REFTBL2(L,S,2.0)	FALSE	PROCEED
STEP 3:	CLAUSE(L,S,OR)	TRUE	N=4.0 J=1
	INRMP(L,S,OR)	FALSE	NONE
	REFTBL2(L,S,4.0)	FALSE	PROCEED
STEP 4:	OUTMP(L,S)	TRUE	NONE
	[SELDOM(1),L,S]=[OMRCD,L,S]	TRUE	CLEAR ALL TUPLES OF REFTB6 WHERE THE DOMAIN NAME ≠ 'EMP#'. SET REFTB20 = REFTB6.
STEP 5:	REFTBL2(L,S,1) AND REFTBL3(L,S,2.0,2.9)	FALSE	NONE
	REFTBL2(L,S,3)	FALSE	PROCEED

STEP 6:	REFTBL2(L,S,5)	FALSE	PROCEED
STEP 7:	COMNTAB2(L,S,5,20)	TRUE	ENTTB2(L,S,Q.C.T.,5,20) RESTRCT1(L,S) RESTRCT2(L,S)

PROCEDURE 3:

	TEST	*RESULT*	*ACTION*
STEP 1:	OUTMP(L,S)	TRUE	NONE
	CAD2(STAFFINFO,EMP#)	FALSE	NONE
	CAD2(EMPIDSAL,EMP#)	FALSE	PROCEED
STEP 2:	CLAUSE(L,S,WHERE)	TRUE	NONE
	NOT(INRMP(L,S,WHERE))	TRUE	NONE
	CLSOPR(L,S,WHERE,=) OR CLSOPR(L,S,WHERE,≠)	FALSE	MARK1=0 N=1
	CAD2(STAFFINFO,EXPER)	FALSE	NONE
	MARK1≠N and MARK1≠0	FALSE	PROCEED
STEP 3:	CLAUSE(L,S,AND)	TRUE	I=1
	NOT(INRMP(L,S,AND))	TRUE	NONE
	CLSOPR(L,S,AND,=) OR CLSOPR(L,S,AND,≠)	FALSE	MARK1=0 N=1
	CAD2(STAFFINFO,AGE)	FALSE	NONE
	MARK1≠N AND MARK1 ≠ 0	FALSE	PROCEED
STEP 4:	CLAUSE(L,S,OR)	TRUE	I=1
	NOT(INRMP(L,S,OR))	TRUE	NONE
	CLSOPR(L,S,OR,=) OR CLSOPR(L,S,OR,≠)	TRUE	NONE
	DOMVAR(L,S,OR)	FALSE	NONE
	CAD2(STAFFINFO,RACE)	FALSE	PROCEED
STEP 5:	FUNCTN(L,S,SELECT)	FALSE	NONE
	ARITHOP(L,S,SELECT)	FALSE	PROCEED
STEP 6:	FUNCTYPE(L,S,1,WHERE,SET)	FALSE	PROCEED
STEP 7:	CLAUSE(L,S,AND)	TRUE	I=1
	FUNCTYPE(L,S,1,AND,SET)	FALSE	PROCEED
STEP 8:	CLAUSE(L,S,OR)	TRUE	I=1
	FUNCTYPE(L,S,1,OR,SET)	FALSE	PROCEED

L2,S1 PROCESSING RESULTS:

```
SELECT  EMP#
FROM    STAFFINFO
WHERE   EXPER > 1
AND     AGE ≤ 36
OR      RACE = 'A'
```

STAGE 6: New Target Database Query

```
SELECT  NAME
FROM    EMPIDENT
WHERE   EMP# =
        SELECT    EMP#
        FROM      EMPIDSAL
        WHERE     EMP# =
                  SELECT    EMP#
                  FROM      STAFFINFO
                  WHERE     EXPER > 1
                  AND       AGE ≤ 36
                  OR        RACE = 'A'
        AND       SALARY ≥  9000
```

CONCLUSION AND FUTURE RESEARCH

Although the data independent property of a well-designed DBMS allows many types of database organization and restructuring at both logical and physical levels without causing changes to the application programs, there are situations where modification of application programs is unavoidable and is more beneficial in terms of system efficiency than using the mapping software associated with the schema and the subschema. As illustrated and argued in an earlier section, drastic reorganizations of the physical database, changes of the semantic associations among data entities and addition and deletion of semantic associations and entities often entail changes to the schema, the subschema and therefore the application programs of a DBMS.

The existing data sublanguages, SEQUEL is an example, depend heavily upon the data model and the data submodel used. The sublanguage statements make direct references to the domain names, entity relation and association relation names and, to a certain degree though different from model to model, contain access path specifications to the data to be searched or processed. Thus, most changes made in a subschema would immediately terminate the life of application programs. To really increase the life span of application programs, it is necessary to make the data sublanguage more independent of data models and submodels. How we can achieve this in language design remains to be investigated.

At this stage of our investigation, we have worked out the algorithm for translating limited yet general language patterns in SEQUEL to account for the database changes introduced by CONVERT operators. This algorithm is described in four procedures written in PL/1 like statements. These procedures have been tested based on hand simulation of a large number of queries running against target databases produced by different sets of CONVERT statements. No actual coding of these procedures have been carried out because we believe that it will not add anything more to what we already can learn from the present work and it is not necessary for meeting our objectives stated in the Introduction at this stage of research. Based on our work and analysis in a somewhat restricted scope, we are quite confident to conclude that it is possible to implement a conversion system to either convert sublanguage statements automatically or provide assistance to the translation analyst in identifying the statements which need to be modified or the possible loss of data when running the old program against the new database. This can be done if the database translation task is carried out orderly by a well-defined database conversion language and the sublanguage used is well structured like SEQUEL. In our work, we found that the data tables in the translation dictionary are adequate for query conversions and the database changes due to all the CONVERT operators can be accounted for by query conversions. The algorithm worked out in this work does not have any general application other than the conversion of a subset of SEQUEL queries. Nevertheless, it is believed that the general methodology and analysis process presented can be useful for future works on other languages even though it is clear that the idiosyncracies of the languages (the sublanguages as well as the data mapping languages) have to be accounted for.

Besides the previously mentioned problem of designing a data language which is independent of the data models and submodels, the following problems seem to be worthwhile investigating. First, the access paths to the desired data in the source database can be very much different from the paths in the target database. In general, there are many access paths to the desired data. The selection of the shortest access path for forming the target query, i.e., the optimation of access path selection, needs to be further researched. Second, the selection of a proper access path that will be consistent with the semantics of the new database is an interesting problem. Any transformation of the source query not only has to preserve the original meaning of the query but also has to incorporate the data conditions based on which the target data files are formed. Some way of proving the correctness of the converted queries based on the semantics of the source queries and the target database would be a task of great importance.

Another problem under our current investigation is based on the following observation. The basic semantic patterns, such as one-one, one-many, many-many associations between/among entities, existing in different database systems are often the same even though different systems may use a different data model for user-database interface. In a given model, the representation of a basic semantic pattern is more or less fixed (e.g., a many-to-many association is represented by two entity relations and an association relation in the relational model,[5] by two owner-coupled sets defined over two record types (for two entities) and a link record type in DBTG's model[3,4] and by two physical databases with proper logical parent pointers to support two logical views of the n:n association in IMS[12]). Since the sublanguage statements for processing the entities and their association depend upon the data representation as defined in the subschema, the access paths to the desired data in each basic semantic pattern should be restricted. The sublanguage patterns for carrying out the access paths can be exhaustively delineated. When a semantic pattern of data has been changed in a database translation, it should be possible to determine first the proper representation of the new data at the subschema level and then the sublanguage patterns which will do the same job as the source language pattern for the source data semantics. Furthermore, it should be possible to determine the pattern of data access in one source language and generate the corresponding pattern of data access in another language once the same data semantic pattern has been identified. Investigation of a generalized program conversion system and of the problem of converting application programs in one database system to that in another system is in progress. Research results along this line of thought are reported in References 27 and 28.

REFERENCES

1. Astrahan, M. M. and D. D. Chamberlin, "Implementation of a Structured English Query Language," *Comm. ACM*, 18, 10, Oct. 1975, pp. 580-588.

2. Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proc. ACM-SIGMOD Workshop on Data Description, Access, and Control,* Ann Arbor, Michigan, May 1-3, 1974.
3. CODASYL, Data Base Task Group Report, April, 1971.
4. CODASYL, Journal of Development, Data Base Facility, IV-4-1 ff, May, 1975.
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Bank," *Comm. ACM,* 13, 6, June 1970, pp. 377-387.
6. Date, C. J., *An Introduction to Data Base Systems,* Second Edition, Addison-Wesley, 1977.
7. Fry, J. P. and E. J. Sibley, "A Data Definition Language Approach to File Translation and File Conversion," University of Michigan, ISDOS Working Paper 44, 1971.
8. Fry, J. P., R. L. Frank, and E. A. Hershey III, "A Developmental Model for Data Translation," ACM SIGFIDET Workshop on Data Description and Access, 1972.
9. Housel, B. C., V. Y. Lum, and N. C. Shu, "Architecture to an Interactive Migration System (AIMS)," *Proc. of ACM SIDFIDET Workshop on Data Description and Control,* 1974.
10. Housel, B. C., D. P. Smith, N. C. Shu, and V. Y. Lum, "DEFINE: A Nonprocedural Data Description Language for Defining Information Easily," *ACM Pacific Regional Conference,* San Francisco, 1975, pp. 62-70.
11. Housel, B. C., "A Unified Approach to Program and Data Conversion," *Proc. of the 3rd International Conference on Very Large Databases,* ACM, N.Y., 1977, pp. 327-335.
12. Information Management System/360 Version 2 General Information Manual, IBM Form No. GH20-0765.
13. Lum, V. Y., N. C. Shu, and B. C. Housel, "Data Translation, PART I: A General Methodology for Data Conversion and Restructuring," IBM Research Report RJ 1525, July, 1975.
14. Merten, A. G. and J. P. Fry, "A Data Description Language Approach to File Translation," *Proceedings of the Workshop on Data Description, Access, and Control,* 1974, pp. 191-205.
15. Navathe, S. B. and J. P. Fry, "Restructuring for Large Databases," *ACM Transactions on Database Systems,* Vol. 1, No. 2, June 1976, pp. 138-158.
16. Ramirez, J. A., N. A. Rin, and N. S. Prywes, "Automatic Generation of Data Conversion Program Using a Data Description Language," *Proceedings of the ACM SIGFIDET Workshop on Data Descriptions, Access, and Control,* May 1974.
17. Reynolds, M. J., "Data Sublanguage Conversions Due to Database Changes," Master's Thesis, University of Florida, 1976.
18. SDDTTG of the CODASYL Systems Committee, "Stored-Data Description and Data Translation: A Model and Language," working paper, 1975.
19. Shu, N. C., B. C. Housel, and V. Y. Lum, "CONVERT: A High Level Translation Definition Language for Data Conversion," *Comm. ACM,* 18, 10, Oct. 1975, pp. 557-567.
20. Sibley, E. H. and R. W. Taylor, "A Data Definition and Mapping Language," *Comm. ACM,* 16, 12, Dec. 1973, pp. 750-759.
21. Sibley, E. H. and A. G. Merten, "Transferability and Translation of Programs and Data," in *Information Systems,* COIN IV, edited by J. T. Tou, 1974, pp. 291-309.
22. Smith, D. C. P., "From a Data Description Point of View," *Proceedings of the ACM's SIGPLAN and SIGIR Interface Meeting,* Nov. 1973.
23. Smith, D. C. P., "A Method for Data Translation Using the Stored Data Definition and Translation TASK Group Languages," ACM SIGFIDET Workshop on Data Description and Access, 1972.
24. Smith, D. C. P., "An Approach to Data Description and Conversion," Ph.D. Dissertation, University of Pennsylvania, 1971.
25. Su, Stanley Y. W., "Application Program Conversion Due to Database Changes," *Proceedings of the 2nd International Conference on Very Large Databases,* Brussels, Sept. 1976, pp. 143-157.
26. Su, Stanley Y. W. and M. J. Reynolds, "Conversion of High-level Sublanguage Queries to Account for Database Changes," Technical Report no. PC7701, Electrical Engineering Dept., University of Florida, May, 1977, p. 59.
27. Su, Stanley Y. W. and B. J. Liu, "A Methodology of Application Program Analysis and Conversion Based on Database Semantics," *Proc. 1977 International Conference on Management of Data,* Toronto, Canada, August 3-5, 1977, pp. 75-87.
28. Su, Stanley Y. W., D. H. Lo, and H. Lam, "Application Program Conversion Due to Semantic Changes," paper to be submitted for publication.
29. Taylor, R. W., "Generalized Data Base Management System Data Structures and Their Mapping to Physical Storage," Ph.D. Dissertation, University of Michigan, 1971.

Guidelines to software conversion

by PAUL OLIVER
Department of the Navy
Washington, DC

BACKGROUND

Data processing installations preparing to undergo a system transition are faced with several difficulties; prominent among these being the transition of the current workload to a new system.

A report by the General Accounting Office[1] published in September 1977, states that the annual Federal Government cost of modifying computer programs to enable them to execute correctly on a computer different from the one for which they were originally devised is estimated at more than $450 million. Comparable industry-wide figures are not available, but it is reasonable to assume that the overall cost of software conversion is significant. Furthermore, this is a non-productive cost; conversion per se results in no direct improvement in an organization's ability to fulfill its mission.

Research and development efforts are under way at several universities and research laboratories to determine ways and means of producing portable software, i.e., software which is machine and configuration independent over a set of computer installations (e.g., see References 2 and 3). At the same time, industry is reacting to the problem in a variety of ways, including a softening of architectual differences (e.g., there are some half dozen IBM 370 "derivatives") and an improvement in emulation capabilities. Until such efforts bear practical fruits, data processing organizations will periodically be faced with the prospect of a software conversion effort. Such an effort is invariably faced with distaste and apprehension.

These are several reasons why system conversion is such a disruptive process. First of all, programmers must be shifted from their regular assignments to the conversion task. This is true whether or not an outside contractor is used to assist in the conversion. Proper conversion requires documentation, and old documentation is often found to be inadequate, even in well managed installations. Aggravating this condition is the fact that the programmers who originally coded the system are frequently no longer with the organization. It is also a sad fact that very few software development specifications include provisions for portability, even though techniques for reducing the impact of system changes on software are known and documented. Finally, conversion often will take place in conjunction with the implementation of a new system, thereby adding to the concomitant disruption.

Conversion is a many-faceted enterprise. An organization preparing itself for a conversion needs to consider not only the conversion process itself but also its management, the technical problems to be faced, the relationship of the conversion costs to the cost evaluation of the new systems being considered, the procurement of conversion support services, cost and time estimates, conversion alternatives (such as emulation), and training of personnel. A complete discussion of all of these topics is beyond the scope of any single paper. We will concentrate therefore on the description of the conversion process itself and the technical difficulties associated with it. The emphasis will be on what needs to be done rather than on how to do it. This emphasis reflects a value judgment on the part of the author; namely, that most of the ignorance regarding conversion has to do with the process itself. All too often organizations mistakenly liken conversion to development, fail to plan and prepare properly, and invariably are overly parsimonious in the allocation of resources to the conversion effort.

Before proceeding, the following definitions are provided in order to set the scope of this paper:

> "*Conversion*: By conversions we mean any change made to a program or system of programs solely for the purpose of enabling such a program or system to execute correctly on a computer different from the one for which they were originally devised. *Translation* refers to a largely automated process of conversion in which the original programs themselves serve as adequate specifications for the new programs to be produced. *Recoding* is similar to translation except that the process is largely manual. *Reprogramming* refers to a conversion which may entail a system redesign (e.g., batch to on-line) but no significant functional redesign. *Redesign* refers to a conversion effort which involves functional redesign and is therefore akin to new development."

CONVERSION PROJECT OVERVIEW

Preparation

The first step in the preparation for conversion is the requirements analysis. A review of the planned differences between existing systems and the converted system is par-

ticularly important if the language dialect being converted to has new language modules (which may be fruitfully used in the converted system), or major changes to input-output modules. It will also be necessary to identify the degree to which the compiler being used differs, in its implementation of the language, from the standard specifications for that language.

Tasks, schedules, resource requirements, and end products must be identified. Schedules and resource requirements are particularly difficult to gauge. It is generally a good idea to obtain contractor support in preparing time and resources estimates, since broad experience in conversion is required in making such estimates. The review of existing programs may reveal some which will require no conversion. It is doubtful that there will be many of these. The rest must be collected together with accompanying files and documentation, and placed in the hands of a conversion team.

Finally, the specifications of system changes must be defined. Data file changes may be required; file and record sizes, field contents, file organization, access keys, sort keys, access methods, storage media, and labeling conventions are all likely candidates for change. The conversion will present an opportunity for some needed system restructuring; programs to be combined, intermediate files eliminated, or sort/merges which may be deleted if the restructuring involves a shift from tape to DASD residence for certain files. Also carefully specified must be any processing logic changes which are necessitated by differing language dialects.

Software tools must be identified and developed. Software must be available to load data, copy programs, create extracted versions of test data, perform data and file conversions, compare tests results for validity, and measure tests for reliability. Procedures must be developed and controls and quality assurance standards must be specified.

Programs must be collected in a uniform way to ensure that the correct version (release) on every program is being converted. The software indicated above may be used to create test libraries and to control this step. A procedure for maintenance change inclusion must be developed, and a reference base for changes is hereby established.

Adequate test data must be prepared which will exercise an acceptable portion of the converted programs. Each program should have a set of known inputs which produces known outputs in order to validate each program. Ideally, unit test data can also be used for system testing.

Finally, all related materials must be collected. This includes program and system documentation (flowcharts, narratives, run-books, data layouts), inventories of files and programs, source listings, program assemblies (listings), and a directory of every item's physical location.

Production

As subsystems become ready for actual conversion the translation process begins. This is true even if systems are to be eventually modified. That is, the conventional wisdom is that program translation (i.e., a one-for-one, or close to it, conversion) should precede any modification. This is done to avoid intermingling, and thereby compounding any translation errors or effects with modification errors or effects. The success of the conversion will be closely related to the adequacy of the controls which are applied during the production stage. Controls must be established for the receipt, handling, and distribution of all materials, for the copying and analysis (to ensure the correctness of the copy process) of program tapes; and for the definition and use of job control language programs.

The translation process itself will be partly automated. Many features of programming languages lend themselves to automatic translation, using commercially available or in-house developed utilities. Unfortunately, a large portion of the input-output coding will have to be hand-translated, and in some cases the process will necessarily have to be closer to modification than translation.

Thorough unit and system tests should follow the translation phase (and will have to be repeated after the modification phase, if any). It is generally advisable to desk-trace the programs in a gross way, i.e., through job control, housekeeping, and initial input statements. A monitor which intercepts and analyzes abnormal terminations would be an extremely useful tool during testing. The monitor should be capable of displaying the instruction causing the abnormal termination, the data being processed at that time, and of providing snapshots of selected data/program areas. Also useful would be a file-compare utility to determine the validity of the outputs produced by the translated program and a monitor which could recognize units of untested code. Once a translated system has been successfully tested any required modifications can take place. These may include system restructuring (combination of common subroutines, sort/merge utilities, etc.), changes in logic, and changes in data files.

Finally, the entire process must be thoroughly and carefully documented. The precise form of the documentation will depend on the installation's standards, but should include at least the following:

Converted source programs
Flowcharts of the converted systems
Listing of all job control language programs used
Standard file labels
File conversion parameters
Operating instructions and technical notes
Unit and system test reports

The following is a step-by-step summary of how a production team should perform the translation of programs:

a. Materials are received by the production team and processed by a Control Section. Each tape is analyzed to ensure readability, and copied to backup tapes. Test data are converted to target machine format. Standard job control code is generated. Task estimates and a schedule for the program translation are created by a resource management system.
b. The source program is converted to the target language

by a multi-step process, using appropriate software tools. It has been found useful to first convert the source code to some intermediate language to permit standardized analysis and manipulation. This is particularly true if the conversion is to involve several source and target languages. The eventual restructured intermediate language program is then converted into properly formatted target language code. The target program listings and other documentation are collected and given to the project manager, who assigns the program to an analyst for completion of the documentation.

c. Corrections are made to the target program, again using appropriate software tools. The program is compiled until all diagnostics are resolved. Two programmers should then desk check every line to verify logical equivalence to the source program.
d. Testing begins with the aid of a cross-reference program, a tool to trap processing exceptions and allow continued processing, and a file compare program to verify output data equivalence. Unexecuted code is identified and carefully desk checked.
e. Unreferenced code is located and identified. Old data and procedure names are replaced where required by new names. The source code is formatted to installation standards to ensure the uniform appearance of all programs.
f. When program translation is completed, system enhancements are applied, and maintenance changes are identified and implemented.
g. The completed programs are returned to the project manager for a quality control check. The material is then processed by a control section and prepared for shipment to the implementation group. Backup copies of programs are stored on tape, along with microfilmed copies of listings for future reference.

Implementation

Implementation of the converted system should not take place until the systems software to be used has reached a satisfactory level of stability (this will be a subjective decision). The conversion manager should anticipate changes (upward) in the resources required to compile and execute the converted programs, and should allow for these changes in order to avoid serious degradation in throughput.

Unit and integration testing should be repeated on converted programs and test data once these are installed on the target system. Once this testing is satisfactorily completed, maintenance changes are applied and tested, problems are corrected, and retesting is performed. This process is repeated until all tests are successfully passed. It should be noted that this inclusion of maintenance changes entails the generation and conversion of production test data to be used in the testing process. Note also that no further changes must be applied to the production programs at this time.

In the meantime, the production database itself is converted and tested, and the operating system control language production stream is generated. Finally, production testing using the final version of programs, data, and control language programs takes place (acceptance testing), followed by an appropriate period of parallel testing.

Software tools

The description of the conversion phases included several references to software tools. A complete inventory of such tools is expensive, and this would be one of the factors which must be taken into consideration when contemplating an in-house conversion, since this inventory would not be of great value after the conversion.

There are several ways in which a software support inventory could be characterized, the simplest of which is according to the three principal conversion stages in which they are used:

a. Preparation: A file contents analyzer, a data extractor and modifier, and a data generator can all be used for creating test data. Utilities will also be required for creating backups of all programs, and for maintaining a current version of the software inventory to be converted and producing statistics (e.g., average program sizes) as required.
b. Production: The production stage will require software to perform source code to intermediate code translation, to analyze and restructure the intermediate code, to perform intermediate code to target code translation, and to translate test data files. Utilities will be needed to generate the operating system control stream and to apply code corrections to translated code. Additionally, software to produce cross-reference listings, to trap and identify exceptions, to identify unexecuted code, and to perform file comparisons will be needed for testing. A de-compiler or depatcher will be needed if the conversion is from an assembler language to a higher level language.
c. Implementation: Software to validate the results of parallel testing, to identify and implement maintenance changes, and to convert the production data base is required.

Additionally, software aids will be useful in the management of the conversion project. As a minimum, software tools are needed for resource management (identify programs and categorize by source and content, estimate resource requirements, and monitor progress), and standards enforcement (format programs to installation standards, replace old names with standard names, etc.).

Managing the conversion project

A conversion project is only slightly different from any other software production project with respect to its management. Careful planning is required, the project must be initiated and, once initiated, it must be controlled. Finally,

there is a completion phase. If there is a significant difference between a conversion project management and a production project management it is one of emphasis; a conversion project requires (and allows for) more discipline and stricter adherence to procedures. If properly executed, a conversion is very much an assembly-line type of operation, where the total effort is broken down into well-defined tasks which are more dependent on experience and strict adherence to procedures than an innovation and ingenuity for their successful completion. This is true partly because of the high degree to which the conversion process can be automated. It should also be noted that many of the ground-rules for software production do not apply to conversion. For example, manpower and time are not generally interchangeable in a software production project, but, within bounds of good taste, they are in a conversion project.

Productivity rates also differ widely between conversion and development. Software development seems to proceed at some 12-20 lines per man-day for general application software, while conversion may proceed at rates as high as 400 lines of code per man-day.

A word of warning regarding project organization—a common mistake is to make the conversion staff a part-time group which participates in conversion activities but whose members continue to report to their parent organizations. This is an ingenious way to make a mess of the conversion, particularly since it will be nearly impossible to attribute responsibility for the mess to any one person.

The specific makeup and size of the conversion staff will vary with the conversion type and magnitude. The following membership, without specific quantities, is suggested for a large-scale project which will require contractor support. Following each staff category is an indication of the role of that category:

a. Project Leader

   (1) Planning
   (2) Project initiation
   (3) Project control
   (4) Project termination

b. Contracting—staff support advising on

   (1) Type of contract
   (2) Necessary clearances
   (3) Terms and conditions
   (4) Scheduling

c. Operations

   (1) Present status and future needs
   (2) Performance specifications
   (3) Scheduling
   (4) Inventory
   (5) Computer resources

d. Systems Programming

   (1) Inventory
   (2) Sizing a job
   (3) Performance specifications

e. Application systems developers

   (1) Inventory
   (2) Sizing
   (3) Performance specifications

f. Support programming—software tools for

   (1) Inventory
   (2) Quality control
   (3) Production
   (4) Testing

g. Material Control

   (1) Quality assurance
   (2) Backup inventory
   (3) Materials transmittal

h. Clerical—supports entire team

i. Analysis & Programming

   (1) Production
   (2) Testing
   (3) Implementation

Figures 1 and 2 suggest the organizational relationships among the staff components, and Appendix A lists the major tasks to be performed in the form of a checklist.

CONVERSION PROBLEMS

General

A conversion project results in a very large volume of material, and this results in control programs. Staffing required for the conversion project will not be needed at the culmination of conversion and will be diverted from ongoing development and maintenance. Furthermore, there will be periods of peak requirements. This will create serious management problems.

Machine time will also encounter periods of peak loads. This, unfortunately, conflicts with growing production due to the cutover of subsystems. Machine time availability takes on an inverse relationship to conversion requirements. In many cases, conversion requirements plus production requirements become more than the total of machine availability. This causes costly delays in the conversion schedule.

Low resources requirements estimates are caused by a lack of understanding of the conversion process. An esti-

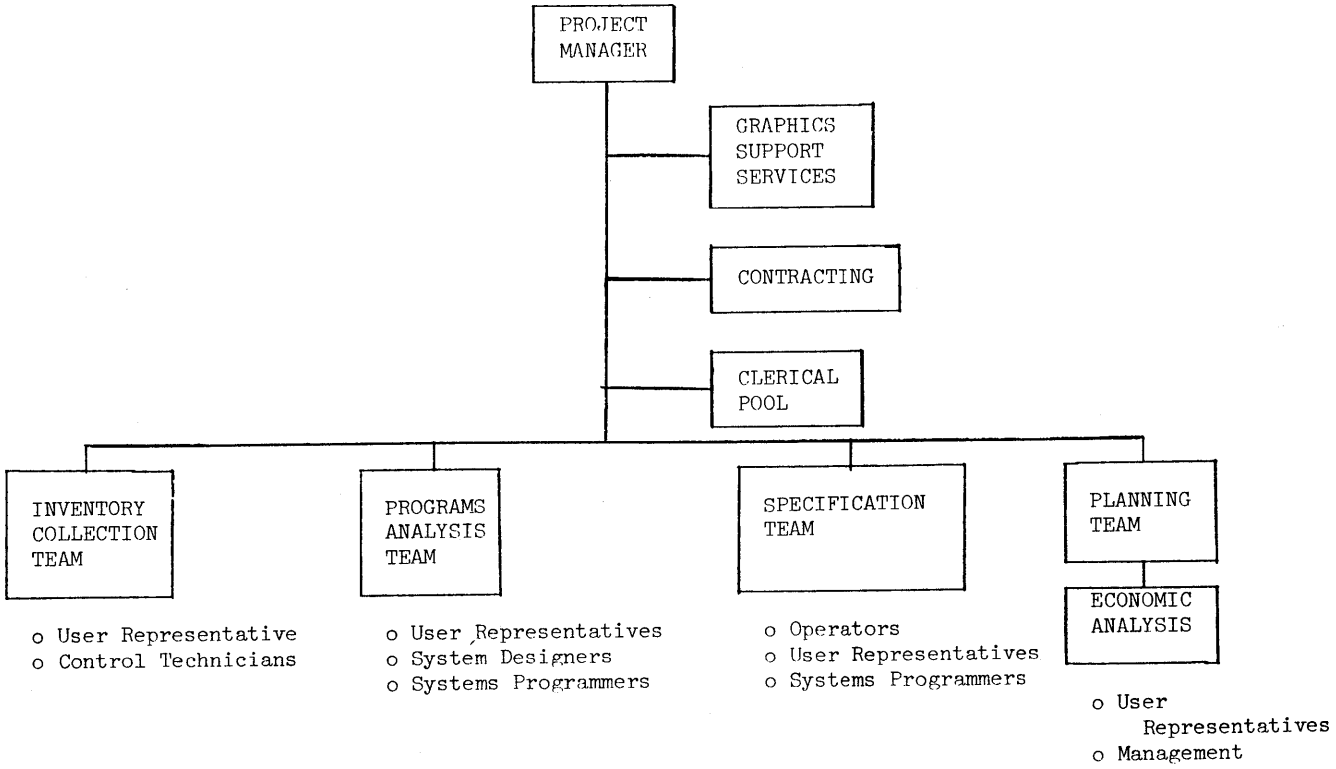


Figure 1—Conversion project organization (Feasibility analysis and planning)

mator may take a few programs, convert them, and attempt to extrapolate the requirements for the whole job. This projection is inaccurate because it represents a straight line relationship (which never really exists) between conversion volume and resource requirements. This mistake is of course the same one which has plagued production time estimators for years. As in software production, size has a special effect on conversion time and costs which render a linear relationship invalid. For example, a competent programmer, properly supported by a set of software tools, can convert 500 lines of COBOL source code per day. This suggests that a 10,000 line program could be converted in 20 days, whereas experience shows that some 36 days would in fact be required, and that this number could go up to 104 days if the conversion is from an assembler language with complex file structures and large file volumes. Another program in estimating is the tendency to exaggerate the extremes by making simple tasks appear too simple, and difficult one too difficult.

Technical

A complete discussion of all the technical problems of conversion is beyond the scope of this paper. What follows is a brief discussion of some of the differences in machine architectures and file structures which can impact conversion.

Computer words vary in the number of characters that they contain causing problems in numerical accuracy and data movement. The accuracy program can be dealt with by increasing arithmetic precision, but this will increase storage requirements as well. The program in data movement is that programs can often refer to the storage of words or characters. When we translate these programs, the net effect is inconsistent treatment of data. A UNIVAC 1108 with a 36-bit word has six 6-bit characters. If we were converting to an IBM S/370, the 1108 program would move six characters in moving a word while the S/370 programs would move four characters in moving a word. To solve the program we have to determine whether or not the code refers to character data. This process often involves lengthy analysis, which requires personnel time, reduces automation, and increases time and costs of conversion.

Machine condition indicators are internal switches that are set for special conditions, e.g., overflow, zero divide, or invalid data. Problems arise when the source machine sets an indicator and the target machine either does not, or does so under slightly different conditions. For example, the 1401 would treat a space as a zero when doing any arithmetic. An IBM 370 will cause an interrupt called a data exception (OC7) when any non-numeric data appear as an operand in an arithmetic statement. This condition must be corrected in the translated program.

The format and the amount of information that is specified to define a file varies among languages. Some languages may

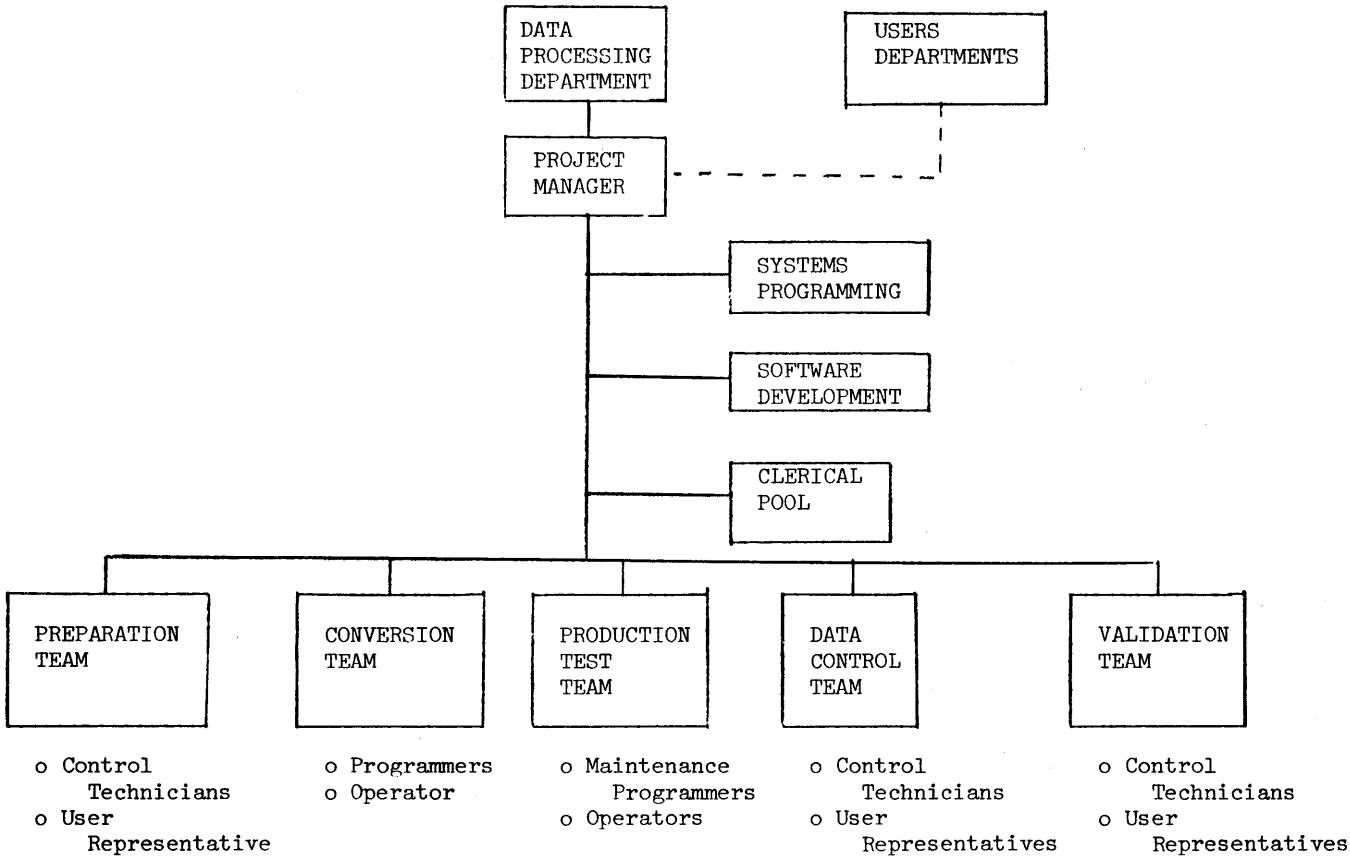


Figure 2—Conversion project organization (Post-planning)

require very little information to be specified, while other languages will require a great deal of information like record length, block size, or labels. Often there is even a wide variance between programs written in the same language because a language will allow, but not require certain specifications, or there will be different ways to define and specify file information in the same language, e.g., standard macros, user macros, or no macros (user-coded I/O).

The specifications of a file may be defined implicitly or explicitly. Explicit definitions are required in a COBOL program. An implicit definition might occur in an assembler program where a file is defined only by its use. Thus, the program and all references to the file must be analyzed to determine the appropriate file definitions. A similar problem also arises in data definition. A data item could be defined explicitly as a character field or not defined at all. All assemblers allow a programmer to specify arithmetic and/or binary operations on any data field. The actual use of a data item defines its implicit attributes. These may differ from their defined attributes in both data type and field length. When there is a discrepancy, an alternate definition must be generated to accommodate the circumstances. This condition is further complicated when implicit usages change because of word size variances.

No "standard" format exists for the recording of variable length records on tape or disk. Some of the ways in which the records are delineated are by delimiting records by special characters to indicate the stopping or starting point of each record, recording the variable length records as fixed length records by padding their length to a maximum consistent length, or inserting byte or word counts at the beginning of each record.

File organization becomes a consideration on any nonsequential file. Generally, indexed files contain pointers to indicate sequencing. Problems arise because variances are encountered in the location of the pointer, the format of the pointer, and the meaning of the pointer (some access methods will use actual track addresses, while others will use a relative record number within the file). The conversion of an indexed file includes getting the disk file onto tape, converting the pointer to the new format, and reloading the file to disk. The manner in which an indexed file is processed usually varies by language and machine. The differences may involve module communication or linkage to the access method for the system. Many direct access applications are developed using an algorithm to compute and determine the location of records on disk. For example, the disk addressing techniques of a source machine may be determined by an algorithm based on bytes per record, records per track, or tracks per cylinder. When converting to a relative addressing technique, or just a disk file with different characteristics, one or all of the above disk parameters could change.

The functions of OPEN and CLOSE in a program will vary. A tape-oriented program may OPEN and CLOSE each tape reel as a separate file; this coding must be eliminated when converting to a concatenated or disk file. Almost all I/O systems have different techniques for handling I/O errors. They also have different error recovery routines. Differnces in these routines must be examined and handled. Communication with error routines must also be dealt with.

Data is frequently represented in different ways on the source and target machines. Often the differences can be easily dealt with, but representation differences can present significant problems. This for example, the BCD (Binary Coded Decimal) and EBCDIC (Extended Binary Coded Decimal) character sets have bit patterns for numbers (0-9) that have a greater binary value than the letters. Several other character sets do not maintain this same relationship and the bit pattern for the numbers has a lower value than for the letters. Conventions and requirements for representing (or not representing) signs on numeric fields often differ between machines produced by the same vendor and always differ between machines produced by different manufacturers. For example, on some machines a negative amount on character numeric fields is indicated by a "a negative zone" and all other numbers are represented by "no zone". On an IBM S/370 there are "−zones," "+zones," and "no zone" fields. On some systems a packed decimal number must always have a sign in the right four bits of the field. The Honeywell 8200 does not require a sign. If a sign is present, it can be from one to four bits long and will appear in the left of the word, not necessarily adjacent to the field. Various Job Control Languages (JCL) are used to relate an application program to the hardware's operating system. Each is usually unique to a computer system. All functions that a program performs with the source operating system must be converted to functions to be performed by the target program and/or operating system. Differences among JCL may exist with regard to inter-module communication. Information (data) is often passed between subroutines within a running program. Sometimes, the data is left in a "common" area of memory by both the calling and the called subroutines. Communicated data must then be isolated and the address of the data passed between modules. Differences may also exist in the roles of the application program and the operating system. In the past, with programs running sequentially in computers, the operating system allowed the program to determine its own hardware usage, file allocation, core allocation, and time requirements. When multiprogramming, these functions are removed from the application programs and supplied by the operating system

In addition to system differences, the differences created by individual organizational programming practices can create significant problems in a conversion effort. In fact, these represent the major factor impacting cost. Sometimes installations using low level languages do not have stringent programming standards. This often results in non-uniform, inconsistent programs. COBOL or PL/I forces a certain degree of standardization. The transformation process must be designed to handle each of the different circumstances and to produce code in a standard high level language. Insufficient, out-of-date or non-existent documentaton is a significant problem for most conversions. The conversion process does not require extensive documentation since we use the source program as the specification for the target program, but system level documentation (or job/file flow) is needed to restructure a system and record layouts are required for file conversions. Systems are often designed with specific hardware/software limitations or user options in mind. Examples are the use of checkpoint/restart for a program that runs too long, using overlays because of core limitations, using a disk file instead of a tape file because of a shortage of tape drives, or running a sort separately instead of as an exit in our application program. Many of these conditions can be changed to improve efficiency when the program or system is converted.

Appendices B and C outline some specific sources of conversion difficulties which may occur in COBOL and FORTRAN programs, respectively. Reference 4 gives an excellent survey of transferability problems of programs written in these and other languages, and Reference 5 suggests techniques to be used in minimizing FORTRAN conversion problems.

CONTRACTING FOR CONVERSION SERVICES

Success in conversion is in large measure dependent on experience and few if any data processing organizations possess this experience. There is little reason why they should—conversion is not an ongoing enterprise. The experience required is not of the "we have three professionals who have gone through a couple of conversions" type. Rather, experience, to be useful, must involve an entire team who has performed enough conversions to become proficient in the techniques and tools to be used, and predictable in its productivity and performance quality. Such experience is most apt to be found in software services or conversion services contractors. Generally, it would be wise for organizations to avail themselves of contractor support for conversion. This is true particularly since, contrary to popular opinion, the software producers are *not* the best qualified people to convert the same software. If anything, they are the worst qualified, since they probably cannot resist the temptation to "improve" the system while converting it.

Contracting methods vary greatly. The best, most sound, contracting is probably done by the U.S. Government under its Federal Procurement Regulations.[6] Useful guidelines can also be obtained from Reference 7. The following are offered as general factors to be considered.

The first step in a procurement is the same as the first step in the conversion planning, namely, a requirements analysis. Once this is done, the contracting personnel can begin to plan for the type of contract which may be required by the specifications, the time required to prepare a request for proposals (RFP) to potential contractors, the time required for negotiations and evaluation of proposals, and the time required to make an award. The availability of funds must also be determined at this time. Our experiences in-

dicate that the procurement may take from 22 to 38 weeks to complete.

The technical staff can ease the burden of the contracting staff by preparing a complete procurement request (PR). This will include a thorough statement of the scope of work (supplies and services, etc.), reporting requirements, property or facilities to be provided to the contractor, and a list of prospective sources of support. Upon receipt of the PR the contracting officer can assemble his procurement staff which, for large or complex procurements may include negotiators, a cost/price analyst, an inspector, legal counsel, and auditors.

A complete discussion of contractor evaluation methods is beyond the scope of this paper, but the principal ones are listed below:

- Cost only—pick low bidder among qualified ones. If the specifications are adequate this is not as bad a method as it is reputed to be.
- Cost plus "desirables"—the basic bid price is modified up or down according to the quality or number of "non-mandatory" items a contractor purposes. This is a superficial way of giving "bonus points" which has little to recommend it in this author's opinion.
- Cost evaluation plus technical evaluation—it is common to attempt to combine "cost points" with "technical points" to determine a winner. This is usually motivated by a desire to substitute subjective judgment for a sound evaluation. One legitimate way of incorporating a technical evaluation into the overall evaluation scheme is to use the results of a technical review to determine a "technical competitive range," to eliminate all bids falling outside this range, and to make the award to the lowest of the remaining bids.

ESTIMATING CONVERSION COSTS

Conversion costs for an in-house conversion (i.e., performed entirely by a staff of programmers from the organization undergoing the conversion) cannot be accurately estimated. The difficulties of estimating production efforts are well-known. Yet, software production is something which programmers do all the time. They do not do conversion all the time. Furthermore, the in-house staff does not have the tools and procedures required, and even if it acquires them it has little or no previous experience in using them.

One can determine reasonable estimates of conversion costs if a contractor is used for the production stage, which is the most costly of the various conversion stages. The various tasks associated with planning and implementation can be itemized, resources required for each of these are estimated, and this results in a cost estimate for these portions of the conversion.

Estimates for the production stage can be derived by reviewing past conversion efforts and prices bid on these efforts. The Federal COBOL Compiler Testing Service has compiled a substantial data base of cost and productivity figures. A complete description of the resulting cost model would require an entire paper itself, and will therefore not be presented here. A few figures can however given general indication of potential costs.

- The cost of converting a line of COBOL code (to COBOL) may range from $.40 to $6.00, with $.40-2.00 being a most likely band, and $.65-1.00 being a good average figure for reasonably clean COBOL programs requiring no extensive file restructuring on additional documentation. FORTRAN to FORTRAN conversion costs are similar to COBOL-COBOL costs.
- Data on PL/I to PL/I conversion is scarce, but indications are that such a conversion would be 10-20 percent costlier than a COBOL-COBOL conversion. Assembler code—COBOL/FORTRAN will cost from $2+ to $8 per source line.
- Productivity figures range from 300-500 lines per man-day in FORTRAN-FORTRAN or COBOL-COBOL down to 20-100 lines per man-day in an assembler-COBOL/FORTRAN conversion.
- Extensive documentation (user guides, narratives, etc.) can cost up to 40 percent of the total per line cost. In-house costs for planning, preparation, and implementation can be up to 50 percent of the total costs.

There are also several qualitative observations which can be made from an analysis of a sizable conversion data base:

- Conversion costs are more dependent on the source machine than on the target machine.
- Knowledge of the application is not critical in performing the conversion; but information regarding the application must be available as required. (Redesign is of course excepted here.)
- Known complexity will not unduly influence costs, but complexity which comes as a surprisse can cause havoc.
- None of the above holds for real-time systems.
- Conversion of data base systems is not well understood.
- Conversion estimates should not be attempted by people who do not have access to a sizable data base of information and who do not do this on a regular basis.
- Cost models consisting of a handful of formulas based on a handful of parameters are worse than useless; they are dangerous because they give the impression of authority.

CONCLUSION

Conversion is a disruptive, largely non-productive process which must nevertheless be faced at some point in the life of a data processing organization. This paper has presented some ideas which might be of assistance to someone contemplating a conversion, the most important of which are

that conversion requires careful planning, and that it should not be attempted without the services of professionals.

REFERENCES

1. General Accounting Office Report of the Congress, FGMSD-77-34, September 15, 1977.
2. Whitten, D. E. and P. A. D. de Maine, "A Machine and Configuration Independent FORTRAN: Portable FORTRAN (PFORTRAN)," Computer Science Department, Pennsylvania State University (1974), University Park, Pennsylvania, 16802.
3. Baird, G. N. and L. A. Johnson, "System for Efficient Program Portability," *AFIPS Conference Proceedings*, AFIPS Press, Vol. 43, 1974.
4. Heiss, Joel E., et al., "Programming for Transferability," International Computer Systems, Inc., 1972, 10801 National Boulevard, Los Angeles, California, 90064.
5. Oliver, P., "Transferability of FORTRAN Benchmarks," National Technical Information Service, ADA039741, Springfield, Va., 1977.
6. Code of Federal Regulation, Title 41, Subtitle A, Chapter 1, Federal Procurement Regulation, The U. S. General Services Administration, 1977.
7. Brandon, D. H., and Sidney Segelstein, *Data Processing Contracts*, Van Nostrand Reinhold, New York, 1976.

APPENDIX A—SYSTEM CONVERSION CHECKLIST

PLANNING
- Collect inventory and define scope of work
- Analyze differences between source and target systems
- Develop conversion plan and schedule
- Estimate resource requirements
- Assign conversion responsibilities

PREPARATION
- Develop or acquire software tools required
- Collect and package programs
- Prepare adequate test data
- Create test output using test data
- Collect related materials

PROJECT CONTROL
- Standardize conversion procedures
- Establish reporting requirements
- Monitor project status
- Assure quality of converted programs

PRODUCTION
- Convert source code
- Code corrections required
- Prepare test job control
- Format programs to standard
- Convert test files

TESTING
- Conduct unit test
- Conduct system test
- Conduct parallel test
- Ensure accuracy of converted programs
- Ensure test data adequacy

INSTALLATION
- Implement maintenance changes
- Prepare production job control
- Convert production files
- Cutover to production

APPENDIX B—COBOL PROGRAMMING PROBLEM SOURCES

GENERAL:
- Requires operator intervention in processing
- Requires operator consol input(s)
- Has checkpoint/restart capabilities
- Contains an interface to database system
- Requires object code patches not incorporated into the current source
- CALLS to an assembler or other non-COBOL language subprogram
- Uses overlays or segmentation

IDENTIFICATION DIVISION ENTRIES:
- Entries out of order with respect to the ANS COBOL Standard
- ENVIRONMENT DIVISION Entries
- FILE CONTROL
- I-O-CONTROL
- DATA DIVISION Entries
- FILE SECTION
- RECORDING MODE
- BLOCK CONTAINS 0
- USAGE IS COMP-1 short precision floating-point data
- COMP-2 long precision floating-point data
- COMP-3 internal decimal data (packed data)
- COMP-4 binary data

WORKING-STORAGE SECTION
- Logic of the program expects certain initial values when data has not been initialized
- REDEFINES
- OCCURS DEPENDING ON
- Bit level data fields (non character aligned)
- Logical switches
- Logical masks
- Floating point literals
- Floating point fields
- Signed zero
- Unsigned numeric fields used in computations
- INDEX
- Subscripts
- Sort description SD-names

LINKAGE SECTION
- Linkage entries

COMMUNICATION SECTION
- Communication description CD-names

REPORT SECTION
- Report writer description RD-names

PROCEDURE DIVISION ENTRIES
- Program logic is sensitive to numeric precision
- Program logic dependent on the collating sequence, i.e., blank
- Logic dependent on HIGH VALUE or LOW VALUE
- Logic dependent on rounding or truncation of numeric results

Logical shifts or bit manipulating
ALTER
CLOSE
COMPUTE
EXAMINE
GO TO DEPENDING
Certain MOVE statements
OPEN
PERFORM (but not PERFORM . . . THRU . . .)
SEARCH
COPY
TRANSFORM
WRITE
SORT
LABEL

An assessment of the technology for data- and program-related conversion*

by JAMES P. FRY
University of Michigan
Ann Arbor, Michigan

EDWARD BIRSS
University of California
Livermore, California

PETER DRESSEN
Honeywell Information Systems
Phoenix, Arizona

NANCY GOGUEN
Bell Laboratory
Piscataway, New Jersey

MICHAEL KAPLAN
Bell Laboratory
Piscataway, New Jersey

EUGENE LOWENTHAL
MRI Systems Corporation
Austin, Texas

VINCENT LUM
IBM Research
San Jose, California

ROBERT MARION
Defense Communication Agency
Reston, Virginia

SHAMKANT NAVATHE
New York University
New York, New York

STEVEN SCHINDLER
University of Michigan
Ann Arbor Michigan

ARIE SHOSHANI
University of California
Berkeley, California

STANLEY SU
University of Florida
Gainesville, Flordia

DONALD SWARTWOUT
University of Michigan
Ann Arbor, Michigan

ROBERT TAYLOR
IBM Research
San Jose, California

BEATRICE YORMARK
Interactive Systems Corporation
Santa Monica, California

INTRODUCTION

The scope of the conversion problem

There are two factors that make conversion necessary: changes in users' functional requirements and changes in their performance requirements. These changes may necessitate the acquisition of new software and hardware, which in turn, may require changes to the existing data programs. For example, the acquisition of a database management system to replace a file management system requires the integration of the original files into a database system and modification of the programs to interact with it. The replacement of a current DBMS with a new database system to provide additional functionality may require a different way of logically structuring the information (its data model) and a different kind of language interface. The establishment of a communication network between differing systems to implement data sharing will require dynamic (i.e., in real time) conversion of data between different nodes in the sense that the same item will repeatedly undergo conversion as it is needed, or alternatively, it requires conversion of programs when they travel to different nodes to access data there. Even when the acquisition of new software or hardware is not warranted, changes in a users' functional and performance requirements can require data and program conversion.

What makes conversion difficult is the proliferation of data models and levels and styles of DBMS interfaces, internal data representation, and hardware architecture. In this section of the report we will examine the technology that has been developed to perform conversions, analyze the areas which require new or improved techniques, and consider strategies for minimizing the need to convert as well as for streamlining those conversions which must be undertaken. Over the past six years, research and development has primarily centered on the problem of converting in non-dynamic environments. The first part of the second section surveys tools and technology currently available for the conversion of data organization. More recently database program conversion—probably the most difficult part of the conversion problem—has received attention. It is reviewed in the second part of the second section, which considers the research directions being taken by current research. The final component of the second section analyzes the status of the entire technology. The third section explores the factors affecting conversion, the approaches for reducing the need to convert data and applications programs, and the impact of new software and hardware technologies on conversion. The fourth section summarizes the trends in conversion needs and tools.

* This is a report of the Conversion Technology Panel which met under the aegis of the ACM/NBS Workshop, "Database Directions—The Conversion Problem". A complete report covering the other panels—Management Objectives, User Experience, and Standarization—will be forthcoming from the National Bureau of Standards.

Components of the conversion process

When conversion is necessary, users will be required to extract data from their source environment and restructure them to the form required for the target environment. While the extraction and restructuring may themselves be complex, these processes are frequently complicated further by the undisciplined nature of the source data. For example, data may exist in duplicate or contain numerous errors and multiple inconsistencies. The whole process of extracting it from its source, "cleaning the data," restructuring them to a desired form, and loading them into the targeted environment is generally referred to as *data conversion* or *translation*.

After a data conversion, particularly one involving extensive restructuring, the application programs which process the original data may not run correctly against the new data. If the amount of restructuring has been small, only a simple modification may be required, while extensive rewrite or redesign may be required when restructuring has been extensive. The process of modifying the programs to process the restructured data is referred to as application program conversion or translation. (In this report we do not consider the problem of converting programs not related to a change in data structure.) Let us discuss data conversion and application program conversion a bit further.

Concepts of data conversion

In brief and conceptual terms, the data translation or conversion process can be represented diagrammatically in Figure 1. As shown in this diagram a data translation system generally requires three components: a reader, a restructurer, and a writer. While the capability of each component depends on the individual design of a data translation system, the function of the reader in general is to access data from its source environment to prepare it for further processing. The accomplishment of this process unquestionably depends on the description of the data. Thus a data description language is needed for this purpose. The writer is the functional inverse of the reader, its function being to put the transformed data into the target environment. It too requires a description of the data structure and shares with the reader the need of a data description language. The function of the restructurer is quite different from that of the other two components. This component in general is responsible for the extraction of data from its source or internal forms and restructuring it to a desired format or structure. Usually a translation description language is required to effect this process.

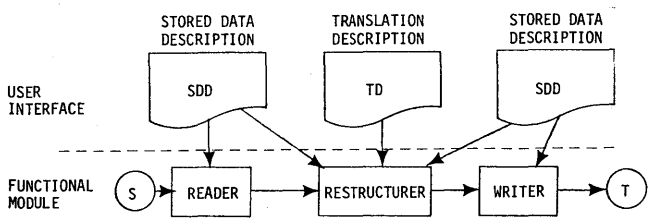


Figure 1—Schematic data conversion process

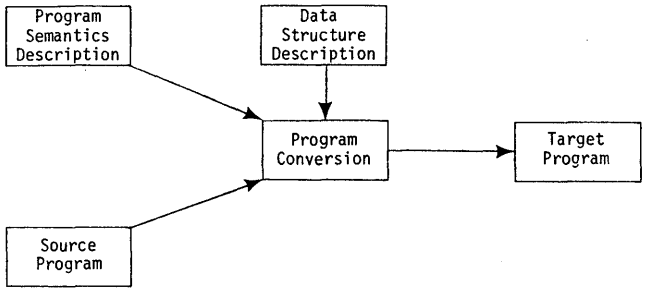


Figure 2—Program translation

In a data conversion system where limited application is intended, the three components may not be distinguishable, nor is the need of the two languages clear. For example, if one wishes to create a conversion system merely to translate EBCDIC characters into ASCII, one can create a simple system with one component and a simple data description language embedded in it. However, if development of a broadly applicable data translation system is the goal, it is clear that one must have a reader and writer capable of accessing and creating data in all kinds of environments and a powerful restructurer capable of all manner of manipulating data and of creating some data as well. Implicit in this application is the need for versatile data translation description languages.

Concepts of database program conversion

Figure 2 represents a general approach to database program conversion or translation. To convert a program it is necessary to determine the functions of the program, and its semantics. Programmers making assumptions about the state of the data may not and currently need not state these assumptions explicitly in their programs. Therefore, it will usually be necessary to provide more information about the semantics of the program than can be extracted from the program text and its documentation. Also needed in the program conversion process is information about the data structure the program originally ran against, the new structure it must run against, and how the two relate. These descriptions could be the same as those used to drive the data translation process. The program, the description of its semantics, and the description of the data conversion are inputs to the program conversion process. It uses them to determine the data originally accessed and how to accomplish the same access in the new structure, and it produces a new program to do this. Currently, the conversion process is accomplished through a combination of manual translation, emulation, and bridge programs. An automatic or semi-automatic program conversion technology is only in the early stages of research.

CONVERSION TECHNOLOGY

In the previous section, we discussed the general concepts of data and program conversion. In this section we shall discuss the technology by which a conversion is achieved.

Since most of the conversion results have been achieved in data conversion, we focus on this aspect and limit our discussion on program conversion to those cases which are the result of a data conversion.

Data conversion technology

Currently, the common approach to data conversion is to develop customized programs for each transfer of data from one environment to another. This approach is inherently expensive; the programs are developed for use only once and their development costs cannot be amortized. Further, the reliability is poor as one is likely to make errors as data is passed from program to program. For a large data conversion effort, this latter effect can indeed be quite unmanageable, as the tracing to the source of an error in the data at a particular point can be extremely complex. An alternative is to search for a broader approach for data conversion with a generalized system. We shall now describe such systems in more detail.

Problem discussion

Data exist in many varied and complex forms; Figure 3, an expanded form of the diagram in Figure 1, indicates some of the transformations that need to take place in a data conversion. This diagram illustrates the capabilities needed by the read and write process in a data conversion system.

The purpose of the unloading of data from its source is to reduce the complex physical structure of the source database to a very simple physical structure, namely a sequential data stream. The source database contains not only the information that interests the user, but also a large amount of control information that is specific to a particular system. This control information is used by the system for such data management functions as overflow chaining, index maintenance, and record blocking. In many systems, when a record is to be deleted it is marked or flagged but not actually deleted. During the unload transformation, all of this system specific information is deleted. Another factor that causes complexity in the unloading process is the frequent use of pointers in a source system. Pointers are used for two basic purposes: (1) to represent relationships that exist between record instances and (2) to implement alternative access paths to the data. During the unload transformation, the second class of pointers may be discarded without loss of information. The first class of pointers, however, maintains information that must be preserved during the transformation.

The purpose of the reformat process is to create a common data form of the source data for the restructuring step in the conversion process. In the case of a simple sequential file, not under DBMS control, it may enter the conversion process at this point. Depending on the design of a system, this step may involve editing (i.e., encoding or recoding of items), limited data extraction, correction, and the like. Since the goal in this step is to create a data structure of the source data without the system-dependent information, the mapping between the input and the output of the reformat process can be considered to be generally one-to-one. While

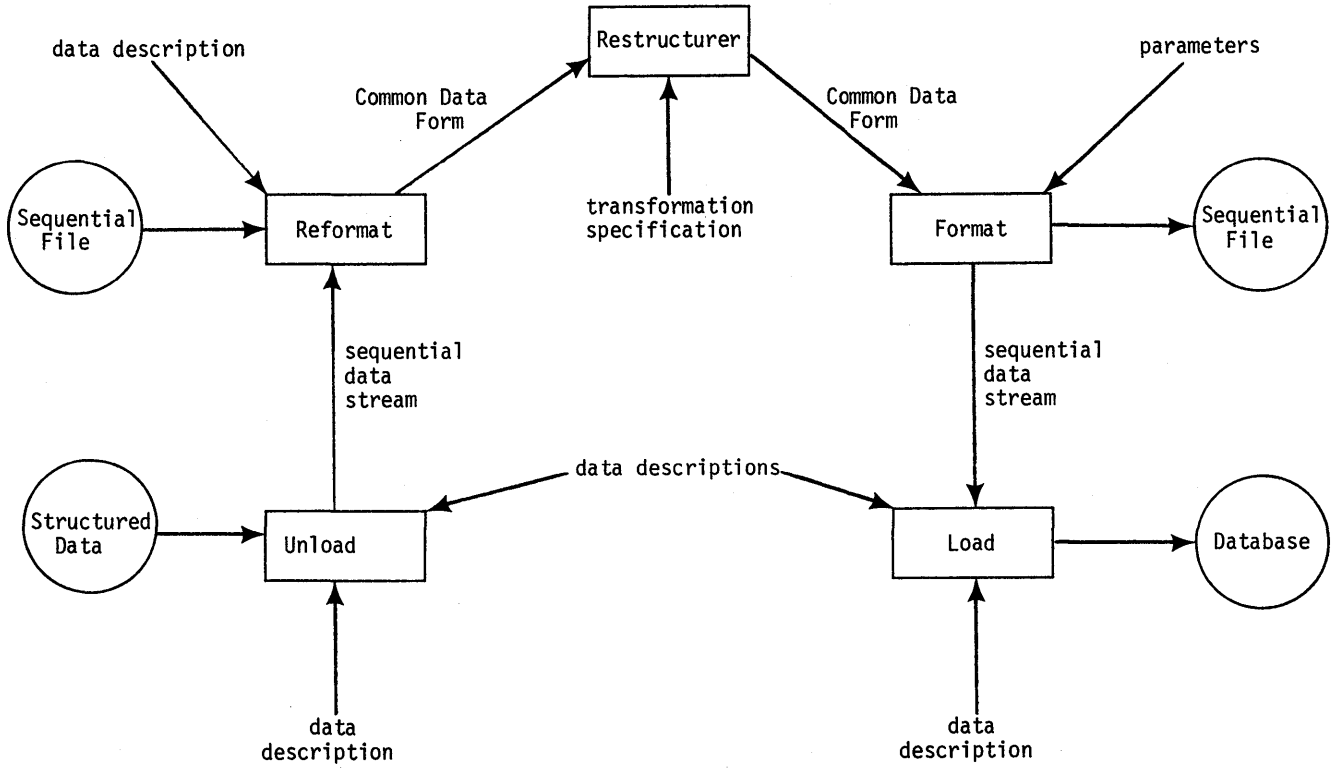


Figure 3—Schematic data conversion process

this step looks simple functionally, its actual specification and implementation can be quite complex. For example, an application program may use the high order bits of a zoned decimal number for its own purposes, knowing that these bits are not used by the system. The specification of nonstandard item encodings is a difficult problem in data conversion.

The load process is the counterpart of the unload process and needs no further clarification. Note that, however, the use of a common data form provides additional benefits, such as easing the portability problem.

The restructuring process is undoubtedly the most complex process of a generalized data conversion system. The languages for this mapping process can be quite different (for example, some procedural and other nonprocedural) and the models used to represent the data in the conversion system are also quite divergent. (For example, some use network structure; others use hierarchical structure). More will be said on this topic later in this section.

Let us now turn to discuss the issue of implementation briefly. Generally there are two techniques. One can build the system using an interpretive approach or the generative approach. In the interpretive approach, the action of the system will be driven by the descriptions written in the system's languages via the general interpreter implemented for the particular system. In the generative approach the data and mapping descriptions are fed into the compiler(s) which generates a set of customized programs executable on a certain machine. The merits of each of these approaches will also be discussed later in this section.

We shall next turn our attention to the tools that have been developed for data conversion. We shall first discuss the tools currently available and then the research and development work in progress.

Available conversion tools

Currently all available tools are limited in capability. Because it is impossible in this short report to provide an exhaustive survey of all the vendor developed conversion tools, we will highlight the spectrum of capabilities available to the user by providing examples from specific vendor shops.

The vendor's repertoire of conversion tools begins at the character encoding level of data conversion with the provision of hardware/firmware options and continues through the software aids for conversion and restructuring of databases.

Depending on a diversity of conditions, the need to develop software tools varies from vendor to vendor. Probably the most prevalent file conversion tool is a COBOL foreign file processing aid. This type of facility allows the direct reading or writing of a particular class of files such as EBCDIC tapes or Honeywell Series 200/2000 files within COBOL. Although a relatively widespread facility, its capabilities are nevertheless limited. For example, some do not handle unlabeled tapes, while others cannot process mixed mode data types. Aside from the work of Bakkom and Behymer,[1] which was aimed toward achieving a general conversion bridge with a particular vendor, to our knowledge there are no vendor supported *generalized* file translation tools.

In contrast to the above file translation tools, tools have been developed that have their main applications in a database environment. One example of a database conversion aid is the I-D-S/II migration aid provided by Honeywell. Because of the large volumes of data involved and the fact that the user cannot afford to shut down his whole data processing shop, a co-existence approach was adopted. The first step is to reformat the I-D-S/I database into the I-D-S/II format, making the necessary data type conversions and pointer mechanism adaptions. This step allows the database to be processed in the I-D-S/II mode, but not optimally. Additional steps in this migration include the generation of the additional I-D-S/II pointer fields (I-D-S/II requires "Prior & Header" chain pointers, which are allocated in step 1 but not filled in) and the restructuring of the I-D-S/II (coexistence) to the more sophisticated capabilities of I-D-S/II.

Some database restructuring tools specific to a particular DBMS have been developed by DBMS users. One example of this type of tool is REORG,[2] a system developed at Bell Laboratories for reorganization of UNIVAC DMS-1100 databases. REORG provides capabilities for logical and physical reorganization of a database using a set of commands independent of DMS-1100 data management commands. A similar capability has been developed at the Allstate Insurance Company.

In addition to the above, there are also software companies and vendors who will do a customized conversion task on a contractual basis.

Data conversion prototypes and models

Over the past seven years a great deal of research on the conversion problem has been performed, with the results summarized in Figure 4. Projects were initiated at the University of Michigan, the University of Pennsylvania, IBM, SDC, and Bell Laboratories, as well as by a task group of the CODASYL Systems Committee. In many cases there was interaction and cross-fertilization between these groups and some consensus on appropriate architectures for data conversion was reached. The individual achievements of these groups is discussed below:

The CODASYL Stored-Data Description and Translation Task Group

In 1970 the CODASYL Systems Committee formed a task group (originally called the Stored Structure Description Language Task Group) to study the problem of data translation. The group presented their initial investigation of the area in the 1970 SIGMOD (then SIGFIDET) annual Workshop in Houston.[3] In 1972 the group was reformulated as the Stored-Data Description and Translation Task Group and a general approach to the development of a detailed model for describing data at all levels of implementation was presented.[4,5] The most recent work of the group specifies the data conversion model and presents an example language

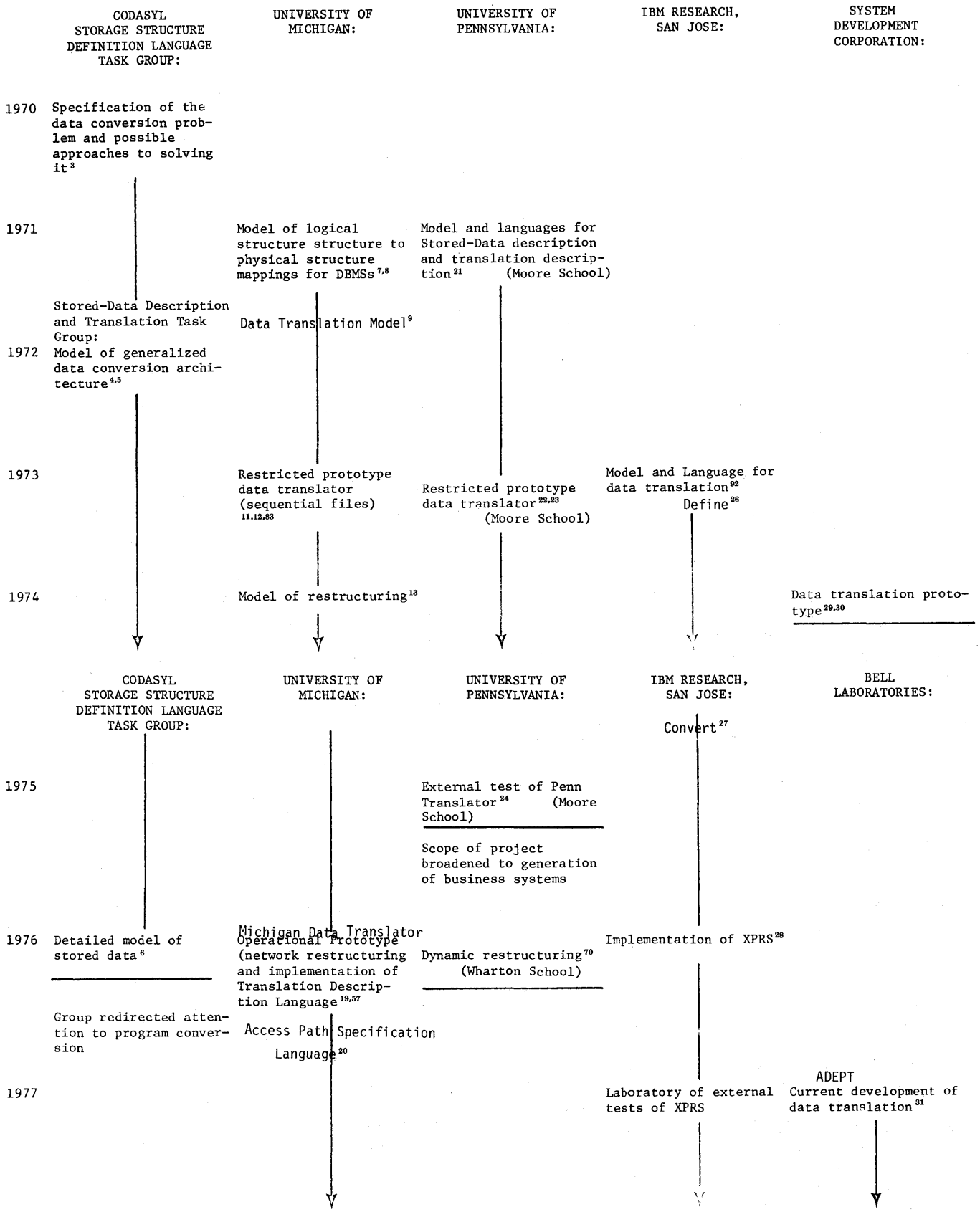


Figure 4—Summary of the development of data conversion models and prototypes

for describing and translating a wide class of logical and physical structures.[6] The stored-data definition language allows data to be described at and distributed to the access path, encoding, and device levels.

The University of Michigan

The nonprocedural approach to stored-data definition set forth by Taylor and Sibley[7,8] provided one of the major foundations for the development at the University of Michigan (see Figure 4) of data translators. In concert with Taylor's language, a model and design for a generalized translation was initiated by Fry, et al.[9]

The translation model was tested in a prototype implementation of the Michigan Data Translator in 1972,[10,11] and the results of the next implementation, Version I, were reported by Merten and Fry.[12]

In 1974, the work of the Data Translation Project of the University of Michigan focused on the database restructuring problem. Navathe and Fry investigated the hierarchical restructuring problem by developing several levels of abstractions, ranging from basic restructuring types to low level operations.[13] Later, Navathe proposed a methodology to accomplish these operations using a relational normal form for the internal representation.[14] Version II of the Michigan Data Translator was designed to perform hierarchical restructuring transformations, but the project did not implement it. Instead, the research was directed into the complex problem of restructuring network type databases. To address this problem Deppe developed a dynamic data model—the Relational Interface model—which simultaneously allowed a relational and network view of the database.[15] This model formed the basis of the Version IIA design and implementation of generalized restructuring capabilities.[16–18] Another component necessary for the development of a restructurer was the formulation of a language in which to express the source to target data transformations. This language, termed Translation Definition Language (TDL), evolved through each translator version beginning with a source-to-target data item "equate list" in the Version I Translator to the network restructuring specifications of Version IIA. While the initial version of the TDL was quite simplistic, the current version, the Access Path Specification Language,[19,20] provides powerful capabilities for transforming network databases.

The University of Pennsylvania

Concurrent with the work at the University of Michigan, Smith at the University of Pennsylvania (see Figure 4), also took a data description approach and developed a stored-data definition language (SDDL) for defining storage of data on secondary storage devices, and a translation description language (TDL)[5,21] and three levels of database structures, the logical, storage, and physical, are described using the SDDL. In order to describe the source-to-target data mappings a first order calculus language was used. Following from this work, Ramirez[22,23] implemented a language-driven "generative" translator which created PL/I programs to perform the conversion. One of the first reports on the utilization of generalization translation tools was provided by Winters and Dickey.[24] Using the translator developed by Ramirez, they installed it on their system, and applied it to conversion IBM 7080 files.

IBM Research, San Jose

In 1973 another major data translation research endeavor was initiated at the IBM Research Laboratory in San Jose, California. Researchers in this project—initially Housel, Lum, and Shu, later joined by Ghosh and Taylor—adopted the general model as specified in Figure 1 but made several innovations. First, in the belief that programmers know well the structure of the data in a buffer being passed from a DBMS to the application program, the group concentrated its effort on designing a data description language appropriate for describing data at this stage. Second, regardless of the data model underlying any DBMS, the data structure at the time it appears in the buffer of an application program will be hierarchical. The general architecture, methodology, and languages reflecting these beliefs is reported in Lum et al.[25]

In addition, the group in San Jose felt that, while it is desirable to have a file with homogeneous record types, it is a fact of life that many of today's data are still in COBOL files in which multiple record types frequently exist within the same file. As a result the group concentrated on designing a data description language which can describe not only hierarchical records (in which a relational structure is a special case) but also most of the commonly used sequential file structures. This language, DEFINE, is described by Housel et al.[26]

The philosophy of restructuring hierarchies is further reflected in the development of the translation definition language CONVERT, as reported by Shu et al.[27] This language, algebraic in structure, consists of a dozen operators, each of which restructures one or more hierarchical files into another file. The language possesses the capability of selecting records and record components, combining data from different files, built-in functions (e.g., SUM and COUNT), and the ability to create fields and vary selection on the basis of a record's content (a CASE statement).

A symmetric process occurs at the output end of the translation system. Sequential files are created to match the need of the target loading facility. The specification of this structure is again made in DEFINE.

A prototype implementation, originally called EXPRESS but renamed XPRS, is reported in Reference 28.

System Development Corporation

Another restructuring project reported by Shoshani[29,30] was performed at The System Development Corporation in 1974-1975. In order to avoid the complexities of storage structure specification (i.e., indexes, pointer chains, inverted tables, and the like) they chose to use existing facilities of the systems involved. In particular they advocated the use of query and load (generate) facilities of database management systems. However, when such facilities do not exist, reformatters from the source (e.g., index sequential file) to a standard form and from the standard form to same output file had to be used. Given that databases can be

reformatted to and from a standard form, they concentrated on the problem of logical restructuring of hierarchical data bases in this form.

The language used in the above project for specifying the restructuring functions (called CDTL—Common Data Translation Language) was designed to be conceptually simple. For the most part it provides functions for specifying a mapping from a single field (or combination of fields) of the source to a single field of the target. For example, while a DIRECT would specify a one-to-one mapping of source items to target items, a REPEAT would specify the repetition of a source item for all instances of a lower level in the target hierarchy. In both cases only the source and target fields need to be mentioned as parameters. In addition there are more global operations, such as the INVERSION operator, which causes parent/dependent record relationships to be reversed. The system also supported extensive field restructuring operators, where individual field values could be manipulated according to prescribed language specifications. Since most of these operators are local, there is the possibility that they could be used in combinations that do not make sense globally. Therefore a further component of the system was built to perform "semantic analysis," which checks for possible inconsistencies before proceeding to generate the target database.

Bell Laboratories

The Bell Labs data translation system ADAPT (A DAta Parsing and Transformation system), currently under development, is a generalized translation system driven by two high-level languages.[31] One language is used to describe the physical and logical format and structure of the data and to provide various tests and computations to be used while parsing the source data and generating the target data. The second language is used to describe the transformations which are to be applied to the source data to produce the target data. Extensive validation criteria can be specified to apply to the source and target data.

Two processing paths are available within the ADAPT system: a file translation path and a database translation path (see Figure 3). A separate path for file translation is provided in response to real-world considerations. Many types of conversion do not require the capabilities and associated high overhead involved in using a database translation path.

Related work

Additional research effort is being devoted to the development and acceptance of a standard interchange form. An interchange form would increase the sharing of databases and provide a basis for development of generalized data translators. The Energy Research and Development Administration (ERDA) has been supporting the Interlaboratory Working Group for Data Exchange (IWGDE) in effort to develop a proposed data interchange form. The proposed interchange form[32] has been used by several ERDA laboratories for transporting data between the laboratories. Additional work on development of interchange forms has been pursued by the Database Systems Research Group at the University of Michigan.[33]

Navathe[34] has recently reported a technique for analyzing the logical and physical structure of databases with a view to facilitating the restructuring specification. Data relationships are divided into identifying and nonidentifying types in order to draw an explicit schema diagram. The physical implementation of the relationships in the schema diagram is represented by means of a schema realization diagram. These diagrammatic representations of the source and target databases could prove to be very useful to a restructuring user.

Application program conversion

So far we have concentrated on the data aspects of the conversion problem; it is necessary to deal as well with the problems of converting the application programs which operate on the databases. Program conversion, in general, may be motivated by many different circumstances, such as hardware migration, new processing requirements, or a decision to adopt a new programming language. Considerable effort has been devoted to special tools such as those to assist migration among different vendor's COBOL compilers, and general purpose "decompilers" to have been developed to translate assembly language programs to equivalent software in a high level language. While progress has been made developing special purpose tools for a limited program conversion situation, little progress has been made in obtaining a solution to the general problem of program conversion. With this fact in mind, this section focuses on the modifications to application programs that arise as a consequence of data restructuring/conversion.

Problem statement

There are three types of database changes which can affect application programs:

- alterations to the database physical structure, for example, the format and encoding of data, or the arrangement of items within records.
- changes to the database logical structure—either
  1. the deletion or addition of access paths to accommodate new performance requirements, or
  2. changes to the semantics of data, for example, modification of defined relationships between record types or the addition or deletion of items within records
- migration to a new DBMS, perhaps encompassing a data model and/or data manipulation language different from the one currently in use.

The actual impact of these database changes on application programs is a function of the amount of data independence provided by the Database Management Systems. Data independence and its relationship to the conversion problem are discussed elsewhere.[59] We assume here that data independence is not complete and that therefore some degree of

program conversion is required in response to database schema changes. In fact, whereas most commercial database management systems provide application programs with insulation from a variety of modifications to the physical database, protection from logical changes—particularly at the semantic level—is minimal. Examples of semantic changes that are likely to have a profound effect on application programs include:

- Changes in relationships between record types, such as changing a one-to-many association to a many-to-many association or vice versa.
- Deletion or addition of data items, record types, or record relationships.
- Changing derivable information ("virtual items") to explicit information ("actual items") or vice versa.
- Changes in integrity, authorization or deletion rules.

There are various properties of database application programs that greatly complicate the conversion problem. For instance many database management systems do not require that the record types of interest (or possibly even the database of interest) be declared at compile time in the program; rather these names can be supplied at run time. Consequently at the compile time, incomplete information exists about what data the program acts on. Other troublesome problems occur when programs implicitly use characteristics of the data which have not been explicitly declared (e.g., a COBOL program executes a paragraph exactly ten times because the programmer knows that a certain repeating group only occurs ten times in each record instance). Complexity is introduced whenever a data manipulation language is intricately embedded in a host language such as COBOL. The interdependence between the semantics of the database accesses and the surrounding software greatly complicates the program analysis stage of conversion. Because of these considerations, substantial research has been devoted to alternatives to literal translation of programs. In particular some currently operational tools utilize source program emulation or source data emulation at run time to handle the problem of incomplete specification of semantics and yet still yield the effects of program conversion.

Current approaches

In this section, we discuss two main techniques currently employed in the industry. These techniques are commonly used but unfortunately not documented in the form of publications.

DML statement substitution

The DML statement substitution technique, which can be considered an emulation approach, preserves the semantics of the original code by intercepting individual DML statements calls at execution time, and substituting new DML statement calls which are correct for the new logical structure of the database. Two IBM software examples which provide this type of conversion methodology are (1) the ISAM compatibility interface within VSAM (this allows programs using ISAM calls to operate on VSAM database), and (2) the BOMP/DBOMP emulation interface to IMS. This program conversion approach becomes extremely complicated when the program operates on a complex database structure. Such a situation may require the conversion software to evaluate each DML operation against the source structure to determine status values (e.g., currency) in order to perform the equivalent DML operation on the new database. Necessary for the generalization of this approach is the development of emulation code for the following cases: maintain the run time descriptions and tables for both the original and new database organizations, intercept all original DML calls, and utilize old-new database access path mapping description (human input) and rules to dynamically determine what set of DML operations on the new database are equivalent to each specific operation on the source database.

Although this approach is straightforward in concept, it has several drawbacks. The drawbacks can be categorized as degraded efficiency and restrictiveness. Efficiency is degraded primarily because each source DML statement must be mapped into a target emulation program, which uses the new DBMS to achieve the same results. The increased overhead in program size and/or processing requirements can be significant.

The drawback of restrictiveness comes about because the emulation approach inhibits the utilization of the increased capabilities of the new DBMS and/or data structure through the modeling of the old methods. Additionally dependence upon the old program semantics limits the sets of permissible new data structures must support all of the semantics of the source program if the source program is to continue to execute in the same manner. It should be noted that the rules can be quite complex, even for the limited situation of which the data structure changes preserve semantic equivalence. Therefore, in some instances, just the limited task of determining if a change in data structure (given no change in the data model) will support a set of source programs will be an extensive task.

Bridge program

The second method in use today is sometimes referred to as the Bridge Program Technique. In this technique, the source application program's access requirements are supported by reconstructing from the target database that portion of the source database needed. Data reconstruction is done by means of "bridge programs." The source program is then allowed to operate upon this reconstructed portion of the source database to effect the same results that would occur if the source database were not modified. Of course, a reverse mapping is required to reflect update and each simulated source database segment must be prepared before it is needed by the application program.

This approach suffers from the same types of disadvantages inherent in the emulation approach. Efficiency problems for complex/extensive databases and programs performing extensive data accessing can make this method prohibitively expensive for practical utilization. This technique is generally found as a "specific software package"

developed at a computer installation rather than as a standard vendor supplied package.

Current research

Differing from the DML statement substitution and bridge program techniques, current research aims toward developing more generalized tools to automatically or semi-automatically modify or rewrite application programs. The drawbacks of the existing approaches described above can be avoided by rewriting the application programs which would take advantage of the new structure and semantics of a converted database and by using a general system to do the conversion rather than using ad hoc emulation packages and bridge programs.

Research on application program conversion is still in its infancy. Consequently, there are very few published papers on this subject. A handfull of works are described here in the order of the dates of publication. Mehl and Wang[35] presented a method to intercept and interpret DL/1 statements to account for some order transforms of hierarchical structures in the context of the IMS system. Algorithms involving command substitution rules for various structural changes have been derived to allow the correct execution of the old application programs. This approach works only for a limited number of order transformation of segments in a logical IMS database. Since it is basically an emulation approach, it has the drawbacks discussed in the previous section.

A paper by Su[36] gives a general model of application program conversion as related to database changes resulting from a database transformation. An attempt was made to identify the tasks required for the automatic or semi-automatic conversion of application programs due to database changes. Two main points are stressed in the paper: (1) the need for extensive analysis of an application program including the analysis of program logic, data variable relations, program-subprogram structure, execution profile, etc., and (2) the use of database translation operators to determine what program transformations are required to account for the effects of these operators. The idea of the use of a common language to describe the operations of source queries and the data translation statements is also proposed.

An approach to the transformation of DBTG-like programs in response to a database restructuring was proposed by Schindler.[37] The approach is based on the concept of code templates, which are predefined sequences of host language—DML statements (roughly analogous to assembly language macros). Application programs can be written as nested code templates. The code templates are devised so that each one corresponds to an operator in the relational algebra. An application program is then mapped into a relational expression, transformations are performed on the expression to accommodate the database restructuring, and a new program is generated by mapping the transformed expression back into code templates. This approach suggests that a level of logical data independence may be achieved through current programming technology.

The work by Su and Reynolds[38] studied the problem of high-level sublanguage query conversion using the relational model with SEQUEL[39] as the sublanguage, DEFINE[26] as the data description language and CONVERT[27] as the translation language. Algorithms for rewriting the source query were derived and hand simulated. In this study, query transformation is dictated by the data translation operators which have been applied to the source database. The purpose of this work was to study the effects of the CONVERT operators on high-level queries. Only restricted types of SEQUEL queries were considered. It is clear from this work that a general program conversion system should separate the data model and schema dependent factors from the data model and schema independent factors; and an abstract representation of program semantics and the semantics of data translation operators need to be sought so that data conversions at the logic level (especially the type which changes the database semantics) and the DBMS level can be attempted.

Two independent works carried out about the same time by Su and Liu[40] and Housel[41] take a more general approach to the application program conversion problem. The former work is based on the idea that the same data semantics (a conceptual model) can be modelled externally by various existing data models (relational, hierarchical and network) using different schemas. Application programs are mapped into an abstract representaton which represents program semantics in terms of the primitive operations (called access patterns) that can be performed on data entities and associations. Transformation rules are then applied on the abstract representation based on the types of changes introduced by the data translation operators. The transformed representation is then mapped into another intermediate representation (called access path graphs) which is dictated by the external model and specific schema used for the target database. This representation is then modified by an optimization component and used for the generation of target programs. It is stressed in this work that the semantics of both the source and target database be made explicit to the conversion system and be used as a basis for application program analysis and transformation. The conversion methodology described is for program conversion to account for data conversion at the logical level as well as the DBMS level.

The work by Housel is an extension of the work on application migration undertaken at the IBM San Jose Laboratory. This work uses a common language for specifying the abstract representation of source programs as well as for specifying the data translation operations. The language is a subset of CONVERT plus some of Codd's relational operators. The operators of the language are designed to have simple semantics and convenient algebraic properties to facilitate program transformation. They are designed to handle data manipulation in a general hierarchical structure called a "form" as well as relational tables. In this system, program transformation is dictated by the data mapping operations applied to the source database. It is assumed in the proposed model that the inverse of these data mapping operators exists, i.e., the source database can be reconstructed from the target database by applying some inverse operators on the target database. More precisely, it is assumed that $M'(T)=S$ where S is the source database, T is the target

database, and M is the mapping function. Thus, program conversion is done by substituting the inverse $M'(T)$ into the specification language statements (the abstract representation of the source program) for each reference to the source database. This process is followed by a simplification procedure to simplify the resulting statements (the target abstract representation of the program). It is pointed out by the author that the assumption on the existence of $M'(T)$ restricts the scope of the conversion problem handled by the proposed approach.

Presently, the Database Program Conversion Task Group (DPCTG) of the CODASYL Systems Committee is investigating the application program conversion problem. The group is looking into various aspects of the problems including decompilation of COBOL application programs, semantic changes of databases and their effects on application programs, program conversion techniques and methodologies, etc.

To this date, the work on application program conversion is still very much at the research stage and more progress has to be made before we can start actual implementation. The problems of automatic application program conversion are multitudinous and extremely complex. Current research indicates that program conversion is possible for some types of data conversion, but the complexity of program conversion depends on how drastically the data have been modified. Further research needs to be undertaken to determine what can be done automatically, what can be done semi-automatically, and what cannot be done at all. A fully automatic tool is hard to achieve. Building semi-automatic systems or systems which provide aids for manual conversion would be a more realistic goal.

Current research directions

The current research has uncovered several problems which need to be further investigated before the implementation of a generalized conversion tool can be attempted. The following issues are believed to be important for future research:

Semantic description of database and application programs

Based on the work by Su and Liu[40] and the study of the DPCTG group, it is quite clear that a program conversion system would need more information about the semantic properties of the source and target databases than the information provided by the schemas of the existing DBMS. Semantic information of the databases is an important source for determining the semantics of application programs which is the real bottleneck of the application program conversion problem. Future research needs to be conducted to (1) model and describe the semantics of application program, (2) study the meaningful semantic changes to databases and their effect on application programs, and (3) derive transformation rules for program conversion which account for the meaningful changes. Several existing works on database semantics[42–46] may provide a good basis for future works on this subject.

Equivalency of source and target programs

Data conversion may alter the semantic contents of the source database. A converted application program may or may not perform the identical operation on the target data as the source program on the source data. For example, it may not retrieve, delete, or update the same data as the source program because some records may be deleted and data relation may have been changed in data conversion. It is not clear at all how we can prove, in general, that a target program generated by a conversion system still preserves the original intent of the source program. Naturally, if the source data can be reconstructed from the target data without losing the original data relations and occurrences, we can establish the equivalence relation between the source and target programs based on the same effect they have on the source and target data.

Decompilation

Program conversion via decompilation is a technique whereby a database application program is first transformed into an operationally equivalent higher order language or an abstract representation and then returned to a usable language level in a converted form. The transformation to a higher order language level is a decompilation process and the process of returning the program to a language level appropriate to conventional compilers is a compilation processs. The underlying concept is that the decompilation to a higher order language can produce a functionally equivalent program that does not contain the DBMS, data model and data dependencies that inhibit the conversion process. That is, the decompiled program has the same "intent" while being unrelated to the changed DBMS environmental conditions. The changed environmental conditions should be easily incorporated into the program during the process of compiling the program back into a form appropriate to the new system.

There is some thought among researchers that this would be the preferred method to effect DML/host language program conversion. It should avoid many of the efficiency/restriction drawbacks inherent in current automated methods, while being more cost effective and less error prone than current manual methods (e.g., program rewrite).

One likely disadvantage to this method is that in order to use it to convert existing database application programs the programs may have to first be manually altered to place DML related code in a structured format. This disadvantage is to be expected because of the ambiguity inherent in the organization of DML/host language programs. However, the development of structured programming templates designed for DML related code should provide a means for creating programs that are convertible by the decompilation method. Structured templates might also provide the needed insight toward the development/selection of an appropriate high level language into which programs can be compiled. Some initial concepts of database program templates have been proposed by the University of Michigan.[37]

Conversion aids

A system which provides assistance to conversion analysts would seem to be a practical tool and a feasible task. Given the information about data changes and semantics of the data, a system can be built to analyze application programs to (1) identify and isolate program segments which are affected by the data changes, (2) detect inefficient code in the programs, (3) produce a program execution profile[47] which gives an estimate of the computation time required at different parts of the program, and (4) detect, in some cases, the program code which depends on the programmer's assumption of data values, ordering of records, record or file size, etc. The data obtained in 1 together with some on-line editing and debugging aids would speed up the manual conversion process. The data obtained in 2 and 3 would be useful for producing more efficient target programs and the data obtained in 4 would help the conversion analyst to eliminate the "implicit semantics" in programs which makes the program conversion task (manual or automatic) extremely difficult. To assist the conversion analysts in identifying ramifications of changes to programs a more complete cross-referencing than that usually produced by today's compilers can be extremely helpful. An example of such a product is the Data Correlation and Documentation system produced by PSI-TRAN Corporation. One technique, sometimes used during a conversion process that has been initiated by a database structure change, is to alter the names of effected database items in the DDL only and use errors generated by the compiler to locate program segments which need to be changed. A more complete cross-referencing system would be a much better tool, if it were available.

Optimization of target program

As the result of data conversion, multiple access paths to the same data may occur. This is because redundant data may be introduced or new access paths may be added in the course of data conversion. In this situation, a conversion system will have the choice of selecting a path to generate the target program. The efficiency of the program during execution time may depend on the selection of optimized access path during program conversion. Also, for reasons of achieving generality, some program conversion techniques proposed[40,41] convert small segments of programs or the equivalent of DML statements separately. It is necessary to do a global optimization or simplification to improve the converted program. Techniques for program optimization related to program conversion need to be investigated.

Analysis of prototype conversion systems

This section analyzes the state of the art of generalized data conversion systems. It summarizes what has been shown to be technically feasible and points out what has been learned in the various prototypes. The prototypes have yielded encouraging results, but some weak points have also emerged. Later sections list some questions that remain to be answered and comment on additional features that will be necessary to enhance usability and analyze some implementation issues which can affect the cases where a generalized conversion system can be applied.

Where do we stand

The prototype systems described earlier have been used in a few conversions. While some of these tests were made on "toy files", a few of the tests involved data volumes from which realistic performance estimates can be extrapolated. This section will summarize the major tests that were done with each of the prototypes.

The Penn Translator

The translator developed by Ramirez at the University of Pennsylvania[22,23] processes single sequential files to produce single sequential target files. Facilities exist for redefining the structure of source file records, reformatting and converting accordingly. Conversion of the file can be done selectively using user-defined selection criteria. Block size, record size, and character code set can be changed, and some useful data manipulation can be included.

The translator was used in several test runs on an IBM/370 Model 165. The DDL to generated PL/I code expansion ratio was 1:4, so coding time was reduced.

A further test of the Penn Translator was conducted by Winters and Dickey.[24] An experiment was conducted comparing a conventional conversion effort against the Penn Translator (slightly modified). Two source files stored on IBM 1301 disks under a system written for the IBM 7080 using the AUTOCODER language were converted to two target files suitable for loading into IMS/VS. Much of the data was modified from one internal coding scheme to another. The conversion required multiple source files to multiple target files.

The conventional conversion took several months versus five months for the generalized approach, a productivity improvement of roughly thirty percent. Time for adapting the translator, learning the DDL, and adapting to a new operating system is included in the five month figure. Without these, an estimate of three months was made for the conversion using the generalized approach.

The SDC translator

The translator described in References 29 and 30 was implemented during 1975-1976. The translator could handle single, hierarchical files from any of three local systems—TDMS, a hierarchical system which fully inverts files; DS/2, a system which partially inverts files; and ORBIT, a bibliographic system which maintains keys and abstracts. Databases were converted from TDMS to ORBIT, from TDMS to DS/2, and vice versa, and from sequential files to ORBIT. TDMS files were unloaded using an unload utility. Target databases were loaded by target system load utilities.

The total effort for design and implementation was about three man-years. The system was implemented in assembly language on an IBM/370 Model 168, and occupied about 40 K-bytes, not including buffer space which could be varied. The largest file tested was on the order of five million char-

acters and the total conversion time was about 1 minute of CPU time per 2.5 megabytes of data.

The work was discontinued in 1976.

The Honeywell translator

The prototype file translator developed at Honeywell by Bakkom and Behymer[1] performed file conversions (one file to one file) among files from IBM, Honeywell 6000, Honeywell 2000, Honeywell 8200 sequential and indexed sequential files. Data types of fields could be changed as well as field justification and alignment. New fields could be added to a record and fields could be permuted within a record. File record format (fixed, variable, blocked, etc.) could be changed and a compare utility was available for checking the consistency of files with different field organizations and encodings. Tests of up to 10,000 records were run. Performance of 15 milliseconds per record was typical (Honeywell Series 6000 Model 6080 computer). The prototype has been used in a conversion/benchmark environment but has not been offered commercially.

The Michigan data translator

Version IIB, Release 1.1 of The Michigan Translator was completed for the Defense Communications Agency in October 1977.[48] It offers complete conversion/restructuring facilities for users of Honeywell sequential, ISP, or I-D-S/I files. Up to five source databases of any type may be merged, restructured or otherwise reorganized into as many as five target databases, all within a single translation. Database description is accomplished by minor extensions to existing I-D-S DDL statements. Restructuring specification is easily indicated via a high level language. Tests performed to date included a conversion of a 150,000 record I-D-S/I database with a total elapsed time of 24 hours (500 milliseconds per record). A given translation can be broken off at any point to permit efficient utilization of limited resources and also protect against system failures. The user is provided with the capability of monitoring translation progress in real time.

XPRS

Test cases with the XPRS system have focussed on functionally duplicating real conversions which had been done previously by conventional methods. Several cases have been programmed. In each case at least two input files were involved. Generally there was a requirement to select some instances from one file, match with instances in another file, eliminate some redundant or unwanted data, and build up a new hierarchical structure in the output. In several cases there was a need for conditional actions based on flags within the data. In all cases, the XPRS languages were found to be functionally adequate to replicate the conversion. A productivity gain of at least fifty percent in total analysis, coding, and debugging time was achieved. Test runs were conducted on several thousand records. Performance was deemed adequate in that XPRS can restructure data at least as fast as it can be delivered from direct access storage. No detailed performance comparisons were made comparing XPRS-generated programs with custom written programs.

Questions remaining to be answered

Given that several prototype data translation systems are operational in a laboratory environment, there is a little question concerning the technical feasibility of building generalized systems. The remaining questions pertain to the use of generalized systems in "real world" data conversions involving a wide variety of data structures, very large data volumes, and significant numbers of people. Three major questions to be resolved are:

1. Are the generalized systems functionally complete enough to be used in real conversions, and if not, what will it take to make them functionally complete?
2. Can the people involved in data conversions use the languages? What additional features are necessary to enhance usability?
3. Overall, what is the productivity gain available with the generalized approach?

Within the next year, prototype systems will be exercised on a variety of real-world problems in data translation, and concrete answers to these questions should be available. The systems being further tested for cost-effectiveness are the Michigan Data Translator, the IBM XPRS system, and the Bell Laboratories ADAPT system.

To date, preliminary results have been promising. A significant sample size on which to do analysis of productivity gain should be available at the end of the year of testing.

A number of factors must be taken into account in measuring the cost-effectiveness of the generalized data translator versus the conventional conversion approach. These factors include:

—ease of learning and using the higher level languages which drive the generalized translators;
—availability of functional capability to accomplish real-world data conversion applications within the generalized translators;
—overall machine efficiency;
—correctness of results from the conversion;
—ability to respond in timely fashion to changes in conversion requirements (conversion program "maintenance");
—debugging costs;
—ability to provide "bridge back" of converted data to old applications;
—ability to provide verification of correctness of data conversion;
—capabilities for detection and control of data errors.

The languages used to drive generalized data translators are high-level and non-procedural; they provide a "user-friendly" interface to the translators. Since the languages are high-level, programs written in them have a better chance of being correct. Experience to date with DEFINE and CONVERT, the languages which drive XPRS, has

shown that users can learn these languages within a week; it has also shown that some practice is necessary before users start thinking about their conversion problem in nonprocedural rather than procedural terms.

In early test cases, the languages which drive generalized data translators have been found to be functionally adequate for many common cases. In those cases where a feature is lacking, a "user hook" facility is often provided. However, forcing a user to revert to a programming language "hook" defeats the purpose of the high level approach, and interfacing the hook to the system requires at least some knowledge of system interfaces. Thus it is important that the high level languages cover the vast majority of the cases in order to succeed; otherwise, users will perceive little difference over conventional approaches.

Facilities for detecting and controlling data errors in the generalized systems are very important, and most of the prototypes do not yet do a complete job in this area. However, the generalized packages offer an opportunity for generalized, high-level methods for dealing with data errors during conversion, and it could well be that once these error packages are developed, they will contribute to even larger productivity gains than have been experienced to date.

The high-level language approach to driving generalized translators should provide the ability to respond to changes in conversion requirements with relative ease. Since large conversions often take one or more years, it is not unusual for the target database design to change or for new requirements to be placed on the conversion system. In other words, in a large conversion effort, the programs are not as "one shot" as is commonly believed. In large conversions, the savings in conversion program maintenance could be significant.

Generalized systems can also be used to map target data back to the old source data form, assuming the original conversion was information-preserving. This capability provides a means for verifying the correctness of the data conversion. In addition, this capability can be used as a "bridge back" to allow users to continue to run programs which have not yet been converted against the data in the old format. Using a generalized system in this way allows phased conversion of programs without impacting user needs during the conversion period.

In an environment where a generalized translator is used regularly as a tool for conversion, costs associated with the debugging phase should be decreased through the use of common software modules. It is unusual in the conventional approach for common conversion modules to be developed. Thus each new conversion system requires debugging.

Usability

The usability of generalized data translation systems must also be evaluated. Experience to date indicates that the languages are easy to learn and use. However, it would be wrong to think that these prototypes are mature software products or that they can be used in all conversions. This section discusses some of the unanswered questions with respect to usability of the current data conversion systems.

One question concerns the level of users of the generalized languages. Current prototypes have been used by application specialists and/or members of a database support group. The systems have not yet been used by programmers, and the question remains whether programmers (as opposed to more senior application specialists and analysts) will be able to use the systems productively. There is no negative data on this point; the systems have not been used widely enough.

At present, all the systems require a user to describe explicitly the source data to be accessed by the read step using a special data description language. These data description languages are generally easy to learn and use; they resemble statements in the COBOL Data Division. However, the writing of the description is a manual process which can be tedious because a person may have to describe a file with hundreds of fields. Ideally, a data conversion system should be able to make use of an existing data description, such as those existing in a data dictionary or a system COBOL macro library. As evidenced by the Michigan Data Translator,[19] it is reasonable to expect that such an interface will be available as data conversion systems evolve. It should be pointed out, however, that a data dictionary or COBOL macro library link may not necessarily solve the problem. Data in current systems is not always fully enough defined to be converted. This is especially true with non-database files. In these files, data definition often is embedded in the record structures of the programs, and a full definition depends on a knowledge of the procedural program logic. Even with existing databases, some fields and associations may not be fully defined within the system database description. Thus, the user can expect a certain amount of manual effort in developing data definitions. If existing documentation is incomplete, this can be a time consuming task, though it probably must be done regardless of whether a generalized package is used or not.

Another area where a user may have to expend effort is in the unload step of the data conversion process. The data description languages used to drive the read step have a limited ability to deal with data at a level close to the hardware (e.g., pointers, storage allocation bit maps, etc.). It is generally assumed that a system utility program can be used to unload source data and remove the more complex internal structures. Another alternative is to run the read step on top of an existing access method or database management system with the accessing software removing the more complex, machine dependent structures. These approaches are acceptable in a great many environments, including most COBOL environments, but there may be cases where neither approach will work. For example, a load/unload utility may not exist, or a file with embedded pointers which was accessed directly by an assembly language program might not be under the control of an access method. For these cases, the user is faced with complexity during the unload step. The complexity associated with accessing the data would appear to be a factor for either the conventional methods or for the generalized approach. However, in cases such as those above, some special purpose software may have to be developed. It should be noted that some research[6] has examined the difficulty of extending data description

languages to deal directly with these more complex cases. The conclusion is that providing the data description language with capabilities to deal with more complex data structures greatly complicates the implementation and has an adverse affect on usability. Thus, special purpose unload programs will continue to be required to deal with some files.

Analysis of architectures

This section discusses some of the different approaches that have been taken in implementing the prototype data conversion systems. The objective is to analyze some of the performance and usability issues raised by the prototypes.

Two approaches have been used in the prototypes—generative approach in the Penn Translator and XPRS, and an interpretive approach in the Michigan Data Translator. In the generative approach, a description of the input files, output files, and restructuring operations is fed to a program generator. From these descriptions, special purpose programs are generated to accomplish the described conversion. In both the Penn Translator and XPRS, PL/I is the target language for the generator. The generated PL/I programs are then compiled and run. In the interpretive approach, tables are built from the data and/or restructuring description. These tables are then interpreted to carry out the data conversion.

In data conversion systems, as in other software, an implementation based on interpretation can be expected to run considerably more slowly than one based on generation and compilation. Initial experience with prototype data translators has shown that there is much repetitive work, strategies for which can be decided at program compilation/generation time. Also, there is a good deal of low level data handling, such as item type conversions. Thus, those implementations based largely on an interpretive approach run more slowly, and the ability to vary bindings at run time does not appear to be necessary. Interpretation was chosen in the prototypes for ease of implementation, and in the future it can be expected that a compilation-based approach or a mixture of compilation with interpretation will be the dominant implementation architecture. However, for medium scale databases, the machine requirements of the interpretive data conversion prototypes are not unreasonable, and overall productivity gains are still possible.

Performance measurements with conversion systems based on the generative approach indicate that generalized systems can be quite competitive with customized programs. In one case, the program generated by the data conversion system ran slightly faster than a "customized" program which had been written to do the same job. However, this example could well be the exception and it would be naive to expect this in general. The reason generalized packages can be competitive is that they often have internal algorithms which can plan access strategies to minimize I/O transfer and/or multiple passes over the source data. "Customized" conversion programs written in a conventional programming language often are not carefully optimized, since the expectation is that the programs will be discarded when the conversion is done.

A second architectural difference involves the use of an underlying DBMS or not. In both the Penn Translator and XPRS, the generated PL/I program, then executing, accesses sequential files, performs the restructuring, and writes sequential files. On the other hand, the Michigan Data Translator functions as an application program running on a network structured database management system. Thus the interpreter makes calls to the underlying DBMS to retrieve data during restructuring and puts restructured data into the new database.

The two approaches offer different tradeoffs. For example, the Michigan Data Translator can make use of the existing extraction capabilities of a DBMS and perform partial translations easily. In addition, since it operates directly within the network data model, a user does not have to think of "unloading" data to a file model and then reloading it back; rather, the user describes a network to network restructuring much more directly.

On the other hand, when converting non-database data to a database, the use of an underlying DBMS as part of a data translator implies a second order data conversion problem—the non-database data must be converted into the DBMS of the data conversion system, which may or may not be difficult. It can be difficult, for example, when the data model of the data being converted differs significantly from the data model of the DBMS upon which the conversion system is based. Also, the use of an underlying DBMS may also require more on-line storage, whereas the file oriented conversion systems can be made to run tape-to-tape. This can be important in very large database conversions.

In the future, one can expect that data conversion systems will offer a variety of interfaces to accommodate various kinds of conversion situations. For example, it is possible to interface the "file-oriented" conversion systems to run as application programs on top of existing database management systems. It is also possible to develop "reader programs" to load non-database data into conversion systems based on a DBMS. In addition, more automated interfaces to data dictionary packages can be expected in order to improve usability and obviate the need for multiple data definitions.

One possible performance problem with generalized conversion systems lies in the unload phase. For reasons of usability, generalized conversion systems usually rely on an unload utility program to access the source data, thus isolating the conversion package from highly system specific data. A potential problem with this approach is that the unload package may not make good use of existing access paths or may tend to access the source data in a fashion which assumes that the data has recently been reorganized (with respect to overflow areas, etc.). In cases where the data is badly disorganized, a customized unload program which accessed the data at a lower level might run considerably faster, and for very large databases might be the only feasible way to unload the data. It is not clear how common this case is, and one can usually make the argument that the

"special" unload software could be interfaced to the generalized package. However, from a practical standpoint, the unloading phase on a very large, badly disorganized database is a performance unknown, and more sophisticated unload utilities may have to be developed as part of the generalized packages.

Summary

Detailed performance and productivity figures for major conversions should be available in about one year. Expectations are that machine efficiency of the generalized packages based on a generation/compilation approach will be acceptable (no worse than a factor of 2) when compared with conventional conversion programs. Additional enhancements to improve usability can be expected, especially in the areas of data error detection and control and interfaces to data dictionary software. If the savings in conversion program analysis and coding times—often fifty percent or more—are confirmed, then the generalized conversion systems will be ready for extensive use.

OTHER FACTORS AFFECTING CONVERSION

In this section we look at the conversion problem from two aspects. First we address the question—What can we do today to lessen the impact of a future conversion. Second, we look to the future to see what effects future technology and standards will have on the conversion process.

Lessening the conversion effort

In order to identify guidelines for both reducing the need for conversion and for simplifying conversions which are required, it is necessary to consider the entire application software development cycle. This is because poor application design, poor logical database design, inadequate use of and inappropriate DBMS selection each could lead to an environment which may prematurely require an application upgrade or redesign. This redesign could, in many cases, require a major database conversion effort.

The set of guidelines specified below is not intended as a panacea. Instead, it is meant to make designers aware of strategies which make intelligent use of current technology. It is doubtful that all conversions could be avoided if a project adhered strictly to these proposed guidelines. However, adherence to the principles set forth by these guidelines could certainly reduce the probability of conversion, and more importantly, simplify the conversions that are required.

With respect to application design and implementation, the more the application is shielded from system software and hardware implementation details, the easier it becomes for a conversion to take place. For example, a good sequential access method hides the difference between tapes, disks, and drums from the application programs which use the access method.

The logical database design should be specified with a clear understanding of the information environment. A good logical database design reduces the need to restructure because it actually models the environment it is meant to serve. Introduction of data dependencies in the data structure should, if possible, be kept to a minimum. An analysis of the tradeoffs between system performance and likelihood of conversion should definitely be made.

Selecting the wrong or non-optimal database management system, given the application requirements, is also a key problem which can lead to unnecessary and large conversion efforts. The prospective user of a DBMS should, for example, carefully evaluate the data independence characteristics of a proposed DBMS.

The underlying principle of the guidelines which follow is that decisions can be made at the system design and implementation stages which are crucial to the stability of the applications.

Application design guidelines

Requirements analysis

Many of the decisions affecting the long-term effectiveness of the application system (database design as well as application programs) are made during the requirements analysis stage of system development. Questions such as what are the functional requirements of the application, who the database is to serve, how the database is to be used, what are the possible future uses of the data, and what are the performance constraints of the application answered at this stage. It is essential that the designer understand the information environment as much as possible at the outset in order to lessen the probability that frequent conversions will be necessary.

Requirements analysis should focus on information needs and should minimize constraints being imposed by the physical environment. Imposing physical constraints at this junction is dangerous since it can distort the designer's view of the true objectives of the application system. The influence of the physical environment should be considered secondarily, in order that the designer be fully aware of the resulting compromises to the logical requirements. This is not intended to imply that consideration of the physical environment is unimportant. Indeed, if the physical environment is ignored the effect could be development of a set of requirements that are impossible to meet within existing physical and cost constraints.

Program design guidelines

There are three underlying principles motivating this discussion of application program design. They are:

- design for maintainability
- design for the application
- data independence

Keeping sight of all of these during the design of the application program will lessen conversion effects by rendering the application as free as possible from physical considerations.

Designing for maintainability implies that the application should be written in a high-level language with a syntax that permits good program structure. Structured programming techniques such as top-down program design and implementation should be used throughout. The system should be modular with relatively small, functionally oriented programs. The programs should all be well commented and organized for readability. Design reviews and program walkthroughs also help to expose errors in the overall design and "holes" in the application logic at an early stage. It has been well documented that these steps help in making program modifications a much easier task.

One error which is often made in designing programs in a DBMS environment is to let the capabilities of the DBMS drive the design rather than the application. This design error can yield programs which are unnecessarily dependent upon the features of a specific DBMS. For example, in System 2000 one can use a tree to represent a many-to-many relationship instead of using the LINK feature. The parent/child dichotomy that results is an efficient but arbitrary contrivance that cannot easily be undone later on. The key principle here is to concentrate on what results are desired rather than on the implementation details of achieving these results. Simplicity and generalization of the design will provide a very high level of interface to the application programmer which will, in turn, minimize the total amount of software, provide the greatest degree of portability, maintainability, device independence, and data independence.

Of extreme importance in program design is the notion of data independence, i.e., insulating the application program from the way the data is physically stored.

Layered design

In the area of application design, the motivating factor for mitigating the effects of a conversion is to insulate logical operations from physical operations. One of the concepts applied to achieve this is layered design. That is, designing the application as a series of layers, each of which communicates with the system at a different level of abstraction. One can visualize this as an "onion," with hardware as its core and layers of successively more sophisticated software at the outer layers. The user interacts with the outermost skin of the onion, at the highest level of abstraction.

If application programs are written at the outermost layers of the onion, then these programs are smaller, easier to understand, and therefore easier to modify or convert than programs written at lower layers. For example, introduction of a new mainframe will require conversion of the software which references the particulars of the mainframe. However, since the layers are constructed so that physical machine and device independence is realized above some level, only the software below that level is subject to modification. To the extent that application programs stay at the outermost layers (i.e., above the critical layer) reduced conversion effects can be achieved.

We can thus summarize the goal of program design as follows:

- to provide the highest possible application interface to the program
- to maximize program independence from the characteristics of the mainframe, peripherals, and database organization
- to maximize portability of the application program through the use of high-level languages
- to maintain a clean program/data interface

Programming techniques

The previous sections of this chapter have focused on the design decisions which should be made to alleviate the conversion problem. However, regardless of how noble these goals are, poor implementation decisions can go a long way towards diminishing the returns of a good design. Equally important to intelligent design is a set of programming techniques and standards which prohibit programmers from introducing dependencies in code. For example, a "clever" programmer may introduce a word size dependency in a program by using right and left shifts to effect multiplication and division. Of course, there are no hard and fast safeguards against using tricky coding techniques; an effort must be made to make the programmer conscious of the consequences of this kind of coding. In particular, a programmer should not be allowed to jump across layers of the onion, such as to use an access method to directly read or write databases.

Database design

Perhaps the most costly mistake a designer can make is an error in the database design because it has a direct effect on the information that is derivable and the application programs that are created. Incorrect or unanticipated requirements can lead either to information deficient databases or overly complex and general design. An inadequate logical design has the potential for complex user interfaces or extremely long access times. A poor physical design can lead to high maintenance and performance costs. Unfortunately, database design is still an art at the present time. Two surveys report the results in the area to date. Novak and Fry[49] survey the current logical database design methodology and Chen and Yao[50] review database design in general. The work of Bubenko[51] in the development of the CADIS system and the abstraction and generalization techniques of Smith and Smith[46,52] show promise.

An accurate logical design can still be unnecessarily data dependent. Dependencies are inadvertantly or deliberately introduced in the interest of improving system performance. In essence, "purity" is compromised to gain processing efficiencies. Since optimization is a worthwhile goal, insist-

ing on absolute purity may be unreasonable. However, the database designer should at least be aware of contrivances and therefore be in a position to evaluate the relative effects a design decision may have. Designers should become sensitive to their decisions by asking: "How will the data model be affected by a future change in performance requirements? Have I done a reasonable job in insulating applications from data structure elements that are motivated strictly by performance considerations?"

Some examples of induced data dependencies in logical database design which may impact upon conversion are:

- The use of owner-coupled sets in DBTG to implement performance-oriented index structures or orderings on records.
- Storing physical pointers (or database keys) in an information field of a record.
- Combining segment types (in DL/1) to reduce the amount of I/O required to traverse a database.

DBMS utilization and selection

Selection of a DBMS can have a major impact on conversion requirements. Of importance in evaluating a DBMS is to consider products exhibiting the highest level user interface.

A high level DBMS is characterized by both a powerful set of functions and a high degree of data independence from the point of view of the application. With respect to functions, that is, the DML, the distinction between "high level" and "low level" has traditionally centered on whether the DBMS provides user operations on sets of records (select, retrieve, update, or summarize *all* the records or tuples which satisfy some conditions) or whether one is restricted to record-at-a-time processing ("navigation"). The DBMS with the "high-level" set operation approach is significantly more desirable than the navigational record-by-record approach.

DBMS prospects should evaluate the data independence characteristics of a proposed product. Systems are preferred which support an "external schema" or "subschema" feature which permits the record image in the application program (the user work area) to differ significantly from the database format. However, the subschema concept is only one aspect of data independence. In general, it is necessary to determine in what ways and to what extent the application interface is insulated from performance or internal format options. For instance, will programs have to be modified if:

- a decision is made to add or delete an index?
- the amount of space allocated to an item is increased or decreased?
- chains are replaced by pointer arrays?

Other conversion related questions about DBMS product include the following:

- Are there adequate performance and formatting alternatives? Are there too many (i.e., unproductive or incomprehensible) tuning options? Are there adequate performance measurement techniques and tools to guide the exercise of these choices?
- Does the system automatically convert a populated database when a new format option is selected?
- Aside from tuning, does the DBMS gracefully accommodate at least simple external changes such as adding or deleting a record or item type?
- Are there other useful high level facilities associated with or based on the DBMS, such as a report writer, query processor, data dictionary, transaction monitor, accounting system, payroll system, etc.?
- Is there a utility for translating the database into an "interchange form", i.e., a machine independent, serial stream of characters?
- Is the vendor committed to maintaining the product across new operating system and hardware releases/upgrades? Conversely, is the vendor prepared to support the product in order of released versions of the operating system, so the user will not be forced to upgrade?
- What hardware environments are currently supported and what is the vendor's policy regarding conversion to another manufacturer's mainframe?
- What programming language interfaces are available? Can the same DBMS features be used if there is a migration, say, from COBOL to PL/1?
- How intelligent is the system's technique for organizing data on the media? Specifically, will performance deteriorate at an inordinate rate as updating proceeds? How often will reorganization (cleanup) be required? does the DBMS have a built-in reorganization utility? How does the user determine the optimal time to reorganize?
- Are the language facilities and data modeling facilities of DBMS adequate for the anticipated long term requirements of the enterprise? What is the risk of having to convert to a new DBMS?
- Likewise, are the performance characteristics and internal storage structure limitations adequate to meet the long term requirements (response times, database sizes) of the enterprise?
- Are there facilities to assist the user in converting data from a non-DBMS environment or from another DBMS? For instance, can a database be loaded from one or more user defined files?

Impact of future technologies and standards on conversion

In this section we discuss trends in computer hardware technologies, DBMS software directions, and standards development, and consider their impact on data and program conversion. Our intention is to make the reader aware of what to expect in terms of conversion problems rather than give a complete assessment of future technologies. Therefore, we discuss only technologies and standards that will impact conversion problems.

The first three parts discuss the areas of hardware, soft-

ware, and standards and their impact on conversion in some detail. The last part summarizes the major points of our assessment without going into detailed reasoning.

Hardware and architectural technologies

The cost and performance of processor logic and memory continue to improve at a fast rate. As a result, overhead costs are more acceptable, especially when such costs save people's time and work, and provide user oriented functions that do not require a computer expert. In particular, one can now think about using generalized conversion tools not only when it is required as a result of hardware or software changes, but also as a result of a changing application that requires a new more efficient database organization. What could have been a prohibitive cost for a database conversion in the past, may not be a major factor in the future.

At the same time, the cost/performance improvement contributes to the proliferation of databases and therefore accentuates the need of generalized conversion tools. The most cost effective is the process of accessing and maintaining data, the more data is collected on computers. Improvements in hardware (as well as software) technologies create more need for data and program conversion. In addition, the emergence of new technologies, such as communication networks, add another level of sophistication to the way that data can be organized and used. Distributed databases, where multiple databases (or subsets of databases) may reside on different machines, require tools for the integration and the correlation of data. Invariably, data will need to move from system to system dynamically, possibly moving between different hardware/software systems. In this environment generalized tools for dynamic conversion will become a necessity.

In recent years two promising approaches to data management hardware technologies have been pursued. One is the specialized data management machine and the other is the backend data management machine. As will be explained next, both approaches can help simplify the conversion problem.

The specialized data management hardware is based on the idea of using some kind of an associative memory device, a device that can perform a parallel access to the data based on its content. Such a device eliminates the necessity for organizing the internal structure of a database using indexes, hash tables, pointer structures, etc., which are primarily used for fast access. As a result, the data can be essentially stored in its external logical form, and the data management system can use a high level language based on the logical data structure only. The conversion process is simplified since data is readily available in its logical organization. Referring to the terminology used in previous sections, the functions of unloading and loading of the database can be greatly simplified. Also, no restructuring will be required because of a change in database use, since the physical database organization can be to a large degree independent of its intended use. In addition, the program conversion problem is simplified as a result of the program interfacing to the DBMS using a high level logical language.

Similar benefits can be achieved if backend machines are used. A backend machine is a special processor dedicated to managing storage and database on behalf of a host computer. The primary motive for the backend machine is to off-load the data management function from the host to a specialized machine that can execute this function at much lower cost. From a conversion standpoint, the separation of data management functions from the host promotes the need for a high level logical interface that provides the advantages discussed above. Another advantage is that it is possible to migrate from one host machine to another without effecting the databases and their management, alleviating the need for data conversion if the same backend machine is used with the new host.

Another hardware technology development is mass storage devices, such as the video disks. These devices would make it cost effective to store very large databases, in the order of 10^{10} characters. The problem of converting large databases is compounded by cost considerations of processing this large amount of data. As a result it is likely that these databases will tend to stay in the same environment for longer periods of time. The use of specialized data management machines or dedicated backend machines in conjunction with these mass storage devices can help postpone the need for database conversion.

Finally, we should mention that there is a growing use of minicomputers in supporting data management functions. DBMSs are now available on many minicomputers, and more development is forthcoming. The proliferation of minicomputers which support databases will only increase the needs for generalized conversion tools.

Software development trends

Much of the work over the last years in the data management area have concentrated on techniques that clearly separate the logical structure of the database from its physical organization. This concept, called "data independence" was introduced to emphasize that users need not be exposed to the details of the physical organizations of the database, but only to its logical relationships. This led to the development of data access and manipulation languages that depend on the logical data model only. The effect of this trend is similar to that of using specialized data management machines and backend machines discussed previously; namely, the simplification of the unload and load functions since the interface to the DBMS is provided at the logical level only, and the simplification in program conversion for similar reasons.

At the user end of the spectrum, it seems reasonable to assume that the diversity of data models (network, relational, hierarchies and other views that may be developed in the future) will be required for many more decades. This is especially true since there are problem areas that seem to map more naturally into a certain model. Furthermore, it is often the case that users do not agree on the same model

for a given problem area. Obviously this state of affairs only accentuates the need to generalize conversion tools that can restructure databases from one model to another. Even with the development of large scale associative memories, data structures will likely provide economic rationales for their contrived use. Another possibility is the use of a common underlying data model that can accommodate any of the user views. However, this approach will still require some type of a dynamic conversion process between the common view and each of the possible user views.

Standards development

There is much work and controversy in developing standards for DBMS. Standards that are oriented to determine the nature of the DBMS are hard to bring about even in a highly controlled environment because of previous investment in application software and database development, and because of disagreement. For example, there is still much controversy whether the network model proposed by the CODASYL committee is a proper one. It seems reasonable to assume that there will always be non-standard DBMSS. Further, even if such a standard can be adopted, different DBMS implementations will still exist, resulting in different physical databases for the same logical database. In addition, one can safely assume that restructuring because of application needs will still be necessary, and that changes in the standard itself may require conversion.

A standard that is more likely to be accepted is one that effects only the way of interfacing to a DBMS. In particular, from a conversion standpoint, a standard interchange data form (SIDF) will be most useful. A SDIF is a format not unlike a load format for DBMSs. Any advanced DBMS has a load utility that requires sequential data stream in a prespecified format. If a standard for this format can be agreed upon, and if all DBMSs can load and unload from and to this format, then the need for reformatting (as described earlier) is eliminated. The conversion process can be reduced to essentially restructuring only, given that unload and load are part of the DBMS function. A preliminary proposal for such a standard was developed by the ERDA Inter-Working Group on Data Exchange (IWGDE).[32] However, it is only designed to accommodate hierarchical structures. Consideration is now given to the extension of the standard to accommodate more general structures (i.e., networks and rebations). We believe that there are no technical barriers to the development of a SDIF, and that putting such a standard to use would alleviate a major part of the data conversion process.

Summary

The reasoning for the points summarized below is given in the previous parts of this section. We will only state here our assessment of the impact on conversion problems.

a. Hardware development will increase the need for generalized conversion tools (in particular, proliferation of minicomputers, computer networks, and mass storage devices).
b. The reduction in hardware costs will make conversion costs more acceptable.
c. Special Hardware DBMS machine will simplify the conversion process (in particular, for load, unload functions, and program conversion) because they promote interfacing at the logical level.
d. Software advances will not eliminate the need for conversion but can simplify the conversion process in a similar way as C.
e. Multiplicity of logical models are likely to exist, thus adding to the need of conversion tools between models.
f. Standards will not eliminate the conversion problem. Even if a standard is followed, the implementations would be different. Also, it is likely that non-standard DBMS will always exist.
g. Standards can greatly simplify conversion. In particular, a standard for interchange data form is likely to come about (will simplify load and unload and eliminate reformatting).

What can we expect in the next five years and beyond in the database conversion area? The state-of-the-art has advanced enough to date to give hope for generalized tools. Within the next five years we can expect more generalized conversion systems to become operational, but some additional work will be required for moving them from one environment to another. We can expect to have a standard form developed and agreed upon. It will probably take longer before manufacturers will see the benefit of adopting a standard form and provide load and unload facilities using it. However, we can expect them to provide some conversion tools to convert databases from other systems to their own. It will probably take as much as ten years before a generalized converter is available commercially, with manufacturers adhering to a standard form. Another area of concern is the application program conversion that is required as a result of database conversion. This topic was discussed in detail earlier. This is a difficult technical problem even within one data model, and still requires much research. It is hard to expect that a generalized solution for this problem will be achieved within the next five years. However, this problem will be elevated to a large extent as hardware and software development trends promote interfacing at the logical level.

BIBLIOGRAPHY

1. Bakkom, D. E., and J. A. Behymer, "Implementation of a Prototype Generalized File Translator," *Proc. 1975 ACM SIGMOD International Conf. on Management of Data,* W. F. King (ed.), ACM, N.Y., pp. 99-110.
2. Edelman, J. A., E. E. Jones, Y. S. Liaw, Z. A. Nazif, and D. L. Scheidt, "REORG—A Data Base Reorganizer", Bell Laboratories Internal Technical Report, April, 1976.

3. Storage Structure Definition Language Task Group (SSDLTG) of CODASYL Systems Committee, "Introduction to Storage Structure Definition," (by J. P. Fry); "Informal Definitions for the Development of a Storage Structure Definition Language," (by W. C. McGee); "A Procedural Approach to File Translation," (by J. W. Young, Jr.); "Preliminary Discussion of a General Data to Storage Structure Mapping Language," (by E. H. Sibley and R. W. Taylor), *Proc. 1970 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ed. by E. F. Codd, Houston, Tex., Nov. 1970, pp. 368-80.
4. Fry, J. P., D. C. P. Smith, and R. W. Taylor, "An Approach to Stored-Data Definition and Translation," *Proc. 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ed. by A. L. Dean, Denver, Colo., Nov. 1972, pp. 13-55.
5. Smith, D. C. P., "A Method for Data Translation Using the Stored Data and Definition Task Group Languages," *Proc. of the 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 107-124.
6. The Stored-Data Definition and Translation Task Group, "Stored-Data Description and Data Translation: A Model and Language," *Information Systems* 2,3, 1977, pp. 95-148.
7. Taylor, R. W., "Generalized Data Base Management System Data Structures and Their Mapping to Physical Storage," Ph.D. Dissertation, The University of Michigan, Ann Arbor, Mich., 1971.
8. Sibley, E. H. and R. W. Taylor, "A Data Definition and Mapping Language," *Comm. ACM,* 16,12, Dec. 1973, pp. 750-59.
9. Fry, J. P., R. L. Frank, E. A. Hershey, III, "A Developmental Model for Translation," *Proc. 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* A. L. Dean (ed.), ACM, N.Y., pp. 77-106.
10. "Design Specifications of a Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1972.
11. "Users Manuals for the University of Michigan Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.
12. Merten, A. G., and J. P. Fry, "A Data Description Approach to File Translation," *Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 191-205.
13. Navathe, S. B., and J. P. Fry, "Restructuring for Large Data Bases: Three Levels of Abstraction," *ACM Transactions on Database Systems,* 1,2, ACM, N.Y., 1976, pp. 138-158.
14. Navathe, S. B., and A. G. Merten, "Investigations into the Application of the Relation Model of Data to Data Translation," *Proc. 1975 ACM SIGMOD International Conf. on Management of Data,* W. F. King (ed.), ACM, N.Y., pp. 123-138.
15. Deppe, M. E., "A Relational Interface Model for Database Restructuring," Technical Report 76 DT 3, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.
16. Birss, E., M. Deppe, and J. Fry, "Research and Data Reorganization Capabilities for the Version IIA Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.
17. Birss, E., et al., "Program Logic Manual for the Version IIA Data Translator," Working Paper 76 DT 3.1, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.
18. Bodwin, J., et al., "Data Translator Version IIA Release 1 User Manual," Working Paper 76 DT 3.2, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.
19. Swartwout, D. E., M. E. Deppe, and J. P. Fry, "Operational Software for Restructuring Network Databases", *Proc. of the 1977 National Computer Conference,* Vol. 46, AFIPS Press, Montvale, N.J., pp. 499-508.
20. Swartwout, D., "An Access Path Specification Language for Restructuring Network Databases," *Proc. of the 1977 SIGMOD Conference,* ACM, N.Y., pp. 88-101.
21. Smith, D. C. P., "An Approach to Data Description and Conversion," Ph.D. Dissertation, Moore School Report 72-20, University of Pennsylvania, Philadelphia, Pa., 1972.
22. Ramirez, J. A., "Automatic Generation of Data Conversion Programs Using a Data Description Language (DDL)," Ph.D. dissertation, University Pennsylvania, 1973.
23. Ramirez, J. A., N. A. Rin, and N. S. Prywes, "Automatic Conversion of Data Conversion Programs Using a Data Description Language," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 207-225.
24. Winters, E. W., and A. F. Dickey, "A Business Application of Data Translation," *Proceedings of the 1976 SIGMOD International Conference on Management of Data,* Ed. by J. B. Rothnie, Washington, D.C., June 1977, pp. 189-196.
25. Lum, V. Y., N. C. Shu, and B. C. Housel, "A General Methodology for Data Conversion and Restructuring," *IBM R & D Journal,* Vol. 20, No. 5, 1976, pp. 483-497.
26. Housel, B., D. Smith, N. Shu, and V. Lum, "Define: A Non-Procedural Data Description Language for Defining Information Easily," *Proc.* of 1975 ACM Pacific Conference, San Francisco, CA, April 1975, pp. 62-70.
27. Shu, N. C., B. C. Housel, and V. Y. Lum, "CONVERT: A High-Level Translation Definition Language for Data Conversion," *Comm. ACM* 18,10, 1975, pp. 557-567.
28. Shu, N. C., et al., "Express: A Data Extraction, Processing, and Restructuring System," *Transactions on Database Systems,* 2,2, ACM, N.Y., 1977, pp. 134-174.
29. Shoshani, A., "A Logical-Level Approach to Data Base Conversion," *Proc. 1975 ACM/SIGMOD International Conf. on Management of Data,* ACM, N.Y., pp. 112-122.
30. Shoshani, A., and K. Brandon, "On the Implementation of a Logical Data Base Converter," *Proc. International Conference on Very Large Databases,* ACM, N.Y., 1975, pp. 529-531.
31. Goguen, N. H., and M. M. Kaplen, "An Approach to Generalized Data Translation: The ADAPT System," Bell Telephone Laboratories Internal Report, October 5, 1977.
32. ERDA Interlaboratory Working Group for Data Exchange (IWGDE), Annual Report for Fiscal Year 1976, NTIS LBL-5329.
33. Bakkom, D., et al., "Specifications for a Generalized Reader and Interchange Form," Working Paper 77 DT 6.2, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.
34. Navathe, S. B., "Schema Analysis for Database Restructuring," *Proc. 3rd International Conference on Very Large Databases,* ACM, N.Y., 1977, p. 326.
35. Mehl, J. W. and C. P. Wang, "A Study of Order Transformation of Hierarchical Structures in IMS Data Bases," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 125-140.
36. Su, Stanley Y. W., "Application Program Conversion Due to Database Changes," *Proc. of the 2nd International Conference on VLDB,* Brussels, Sept. 8-10, 1976, pp. 143-157.
37. Schindler, S., "An Approach to Database Application Restructuring," Working Paper 76 ST 2.3, Database Systems Research Group, The University of Michigan, Ann Arbor, Mich. 1976.
38. Su, S. Y. W., and M. J. Reynolds, "Conversion of High-Level Sublanguage Queries to Account for Database Changes," Technical Report No., PC7701, Electrical Engineering Dept., University of Florida, May 1977.
39. Chamberlain, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings of the ACM-SIGMOD Workshop on Data Description, Access and Control,* ACM, N.Y., 1974.
40. Su, S. Y. W., and B. J. Liu, "A Methodology of Application Program Analysis and Conversion Based on Database Semantics," *Proceedings of the International Conference on Management of Data,* 1977, pp. 75-87.
41. Housel, B. C., "A Unified Approach to Program and Data Conversion," *Proceedings of the Third Internation Conference on Very Large Databases,* ACM, N,Y,m 1977. To appear in TODS.
42. Chen, P. P. S., "The Entity-Relationship Model—Towards a Unified View of Data," *Transactions on Database Systems* 1,1, 1976, pp. 9-36.
43. Roussopoulos, N., and J. Mylopoulos, "Using Semantic Networks for Database Management," *Proc. Very Large Database Conference,* Framingham, Mass., Sept. 1975, pp. 144-172.
44. Schmid, H. A., and J. R. Swenson, "On the Semantics of the Relational Model," *Proc., ACM-SIGMOD 1975 Conference,* May 1975, pp. 211-233.
45. Su, Stanley Y. W., and D. H. Lo, "A Multi-level Semantic Data Model," CAASM Project, Technical Report No. 9, Electrical Engineering Dept., University of Florida, June 1976, pp. 1-29.
46. Smith, J. M., and D. C. P. Smith, "Database Abstractions: Aggregation

and Generalization," *ACM Transactions on Database Systems* 2,2, 1977, pp. 105-133.
47. Ingals, D., "The Execution Time Profile as a Programming Tool," in *Design and Optimization of Compilers,* ed. by R. Rustin, Prentice Hall, 1972, pp. 108-128.
48. Kintzer, E., et al., "Michigan Data Translator Version IIB Release 1.1 User Manual," Technical Paper 77 DT 8.1, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.
49. Novak, D. and J. Fry, "The State of the Art of Logical Database Design," *Proceedings of the Fifth Texas Conference on Computing Systems,* IEEE, Long Beach, 1976, pp. 30-39.
50. Chen, P. P., and S. B. Yao, "Design and Performance Tools for Data Base Systems," *Proceedings of the Third International Conference on Very Large Data Bases,* ACM, N.Y., 1977, pp. 3-15.
51. Bubenko, J. A., "IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schema," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* ACM, N.Y., 1977, pp. 62-74.
52. Smith, J. M., and D. C. P. Smith, "Database Abstractions: Aggregation," *Communications of the ACM* 20, 6, 1977, pp. 405-13.
54. Frank, R. L. and K. Yamaguchi, "A Model for a Generalized Data Access Method," *Proc. of the 1974 National Computer Conference,* AFIPS Press, Montvale, N.J., pp. 437-444.
55. Taylor, R. W., "Generalized Data Structures for Data Translation," *Proc. Third Texas Conference on Computing Systems,* Austin, Texas, 1974, pp. 6-3-1.
56. UNIVAC, UNIVAC 1100 Series Data File Converter, Programmer Reference UP-8070, Sperry Rand Corporation, March, 1974.
57. Yamaguchi, K., "An Approach to Data Compatibility, A Generalized Access Method," Ph.D. dissertation, University of Michigan, 1975.
58. Birss, E. W. and J. P. Fry, "Generalized Software for Translating Data," *Proc. of the 1976 National Computer Conference,* Vol. 45, AFIPS Press, Montvale, N.J., pp. 889-899.
59. Date, C. J., *An Introduction Database System,* Addition-Wesley, 1975.
60. SHARE AD-HOC Committee on Universal Languages, "The Problem of Programming Communication with Changing Machines: A Proposed Solution," *Comm. ACM,* Aug., 1958, pp. 12-18.
61. SHARE AD-HOC Committee on Universal Languages, "The Problem of Programming Communication with Changing Machines: A Proposed Solution, Part 2," *Comm. ACM,* Sept. 1958, pp. 9-16.
62. Sibley, E. B. and A. G. Merten, "Transferability and Translation of Programs and Data," *Information Systems, COINS IV,* Plenum Press, N.Y., 1972, pp. 291-301.
63. Yamaguchi, K. and A. G. Merten, "Methodology for Transferring Programs and Data," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 141-156.
64. Housel, B. C., V. Y. Lum, and N. Shu, "Architecture to an Interactive Migration System (AIMS)," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 157-170.
65. Housel, B. C. and M. H. Halstead, "A Methodology for Machine Language Decompilation," *Proc. of the 1974 ACM Annual Conference,* ACM, N.Y., pp. 254-260.
66. Honeywell Information Systems, "Functional Specification Task 609 Database Interface Package," Defense Communications Agency Contract DCA 100-73-C-0055.
67. Kintzer, E., "Translating Database Procedures," *Proc. 1975 ACM National Conference,* ACM, N.Y., pp. 359-62.
68. Dale, A. G., and N. B. Dale, "Schema and Occurrence Structure Transformation in Hierarchical Systems," *Proc. 1976 International Conference on Management of Data,* pp. 157-168.
69. Fry, J. P. and D. Jeris, "Towards a Formulation of Data Reorganization," *Proc. 1974 ACM/SIGMOD Workshop on Data Description, Access and Control,* ed. by R. Rustin, ACM, N.Y., pp. 83-100.
70. Housel, B. C., and N. C. Shu, "A High-Level Data Manipulation Language for Hierarchical Data Structures," *Proc. of the 1976 Conference on Data Abstraction, Definition and Structure,* Salt Lake City, Utah, pp. 155-169.
71. Navathe, S. B., "A Methodology for Generalized Database Restructuring," Ph.D. dissertation, The University of Michigan, 1976.
72. Gerritsen, Rob and Howard Morgan, "Dynamic Restructuring of Databases With Generation Data Structures," *Proc. of the 1976 ACM Conference,* ACM, N.Y., pp. 281-286.
73. Bachman, C. W., "The Evolution of Storage Structures," *Comm. ACM* 15,7, July 1972, pp. 628-34.
74. Lewis, K. B. Driver, and M. Deppe, "A Translation Definition Language for the Version II Translator," Working Paper 809, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.
75. Lewis, K. and J. Fry, "A Comparison of Three Translation Definition Languages," Working Paper DT 5.1, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.
76. Deppe, M. E., K. H. Lewis, and D. E. Swartwout, "Restructuring Network Databases: An Overview," Data Translation Project, Technical Report 76 DT 5, University of Michigan, Ann Arbor, Michigan, 1976.
77. Deppe, M. E. and K. H. Lewis, "Data Translation Definition Language Reference Manual for Version IIA Release 1," Data Translation Project, Working Paper 76 DT 5.2, University of Michigan, Ann Arbor, Michigan, 1976.
78. Swartwout, D. E., A. M. Marine, and D. E. Bakkom, "Partial Restructuring Approach to Data Translation," Data Translation Project, Working Paper 76 DT 8.1, University of Michigan, Ann Arbor, Michigan, 1976.
79. Swartwout, D. E., G. J. Wolfe, and C. E. Burpee, "Translation Definition Language Reference Manual for Version IIA Translator, Release 3," Data Translation Project, Working Paper 77 DT 5.3, University of Michigan, Ann Arbor, Michigan, 1977.
80. Data Translation Project, "Stored-Data Definition Language Reference Manual," University of Michigan, Ann Arbor, Michigan, 1972.
81. Data Translation Project, "Revised Stored-Data Definition Language Reference Manual," University of Michigan, Ann Arbor, Michigan, 1974.
82. Data Translation Project, "University of Michigan Stored-Data Definition Language Reference Manual for Version II Translator," University of Michigan, Ann Arbor, Michigan, 1975.
83. Birss, E. W., and J. P. Fry, "A Comparison of Two Languages for Describing Stored Data," Data Translation Project, Technical Report 76 DT 1, University of Michigan, Ann Arbor, Michigan, 1976.
84. "Functional Design Requirements for a Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1972.
85. "Program Logic Manual for the University of Michigan Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.
86. "Functional Design Requirements of the Version I Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.
87. "Program Logic Manual for the University of Michigan Version I Data Translator," Working Paper 306, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1974.
88. "Design Specifications: Version II Data Translator," Working Paper 307, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.
89. Bodwin, J., et al., "Data Translator Version IIA Release 2 User Manual," Working Paper 76 DT 3.4, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1976.
90. Kintzer, E., et al., "Michigan Data Translator Version IIB Release 1 User Manual," Technical Paper 77 DT 8, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.
91. Burpee, C. E., et al., "Michigan Translator Program Logic Manual Version IIB, Release 1," Working Paper 77 DT 3.7, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.
92. DeSmith, D., and L. Hutchins, "Michigan Data Translator Design Specifications Version IIB," Working Paper 77 DT 3.8, Database Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.
93. Bakkom, D. E., and S. J. Schindler, "Operational Capabilities for Database Conversion and Restructuring," Technical Report 77 DT 6, Database Systems Research Group, The University of Michigan, Ann Arbor, Mich., 1977.
94. Housel, B., V. Lum, and N. Shu, "Architecture to an Interactive Migration Systems (AIMS)," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 157-169.

Distributed data base technology—An interim report of the CODASYL systems committee

by The CODASYL Systems Committee

Members:
O. H. BRAY, *Sperry UNIVAC*
M. E. DEPPE, *Hewlett Packard (General Systems)*
G. C. EVEREST, *University of Minnesota*
J. P. FRY, *University of Michigan*
R. R. HAYWARD, *Department of the Navy*
H. F. HERRE, *Fibreboard Corp. (MIS Dept.)*
S. R. KIMBLETON, *National Bureau of Standards*
J. J. MAHLER, *SYCOR*
B. K. PLAGMAN, *DBD Systems, Inc.*
E. H. SIBLEY, *University of Maryland*
W. H. STIEGER, Chairman, *The Standard Oil Company (Ohio)*
J. A. TILLINGHAST, *MRI Systems Corporation*
P. M. WHITING-O'KEEFE, *SRI Information Science Lab.*
J. W. YOUNG, *The NCR Corporation*

INTRODUCTION

The purpose of this paper is to present to the computing community an interim report (as of January 1978) on the work of the CODASYL Systems Committee in analyzing the relationship between database technology and distributed processing. This paper discusses the following topics:

- The motivations for incorporating one or more data bases into a distributed processing environment.
- The current objectives of the Committee's work.
- The network-oriented and database-oriented components, and their relationships.
- The design choices available to the architect of a distributed database environment.
- The technical and administrative issues which arise in such environments.

The Committee hopes that this paper will evoke thoughtful comments from its readers. These will greatly aid the Committee in the preparation of its final report. Comments should be directed to:

Chairman, CODASYL Systems Committee
Box 1808
Washington, DC 20013

PURPOSE/SCOPE OF COMMITTEE'S CURRENT WORK

In accordance with its charter, the CODASYL Systems Committee, in January of 1977, undertook a two-year task examining the implications of database technology in a distributed environment. The Committee is focusing its attention primarily on the impact of extending database management techniques to distributed processing environments.

The scope of this effort is the establishment of a framework for examining these issues, and the formation of baseline concepts and guidelines which will support the continuing development of distributed database management technology. The results of this work will be published at the end of this two-year effort as a Systems Committee Technical Report.

MOTIVATIONS FOR THE DISTRIBUTED DATABASE ENVIRONMENT

In today's evolving data processing technology, a number of factors have generated a trend from centralization toward distribution of data processing functions. Of major importance to the end user is faster, easier access to the data needed for decision making, a need essentially unchanged from earlier, more centralized environments. A further concern of the end user is reliability and security. These user motivations, coupled with the increasing geographic dispersion of the end users within an organization, are generating pressures on data processing and corporate management to distribute data processing and storage capabilities to the location of data origin and/or use. The major benefits derived by distributing these functions focus on increased data availability to the end user and reduced exposure to total system failure due to hardware/software failure.

Historically, the application of computer technology to the information processing needs of an organization have

tended toward the centralization of data processing and storage facilities. Motivations for such centralization of data processing functions have focused on the following:

- *Economies of Scale*—the cost of processing and storage facilities sufficient to handle the local processing requirements of the end user generally exceeded the combined cost of large, centralized computer facilities and a communications facility to provide user access to the central facility.
- *Control*—security, data integrity, and standards administration in a centralized environment generally were less complex than in a decentralized environment.
- *Availability of Data to the Corporate Level*—Although not fully satisfying the needs of the distributed end user, the centralized environment provided easier generation of and access to summary data.

Today, organizations have grown in size and complexity, the users have become more sophisticated in their information needs, and the geographic locations of the origin and use of data have become increasingly dispersed. As a result, the strategy of centralizing the EDP function may run contrary to the objective of availability of data to the end user. Further, a number of technological factors encourage the distribution of data processing and storage facilities.

These factors include:

- *Lower Cost of Processing and Storage Facilities*—recent advances in computer technology and a rapid decrease in hardware cost have reduced or eliminated the cost advantages experienced through centralization.
- *Cost of Communications*—relative to the price/performance gains achieved in computer hardware, communications cost, in terms of price/performance, have remained fairly stable, further motivating decentralization of data processing and storage facilities in order to minimize communications traffic and associated costs.
- *Improvements in Communications Hardware and Software Technology*—though not as dramatic as witnessed in computer hardware and software technology, certain developments in communications technology are motivating factors in the trend toward distributing the data processing function. For example, the needs for control a motivating factor for centralization, can now be accommodated with facilities to "down load" control, software, standard application software, or special procedures to the local processing nodes from a central or controlling node. Additionally, communications technology today readily accommodates the need to access one or more nodes in a distributed environment from a central or controlling node in order to gather summary data.

In summary, technological and price/performance developments in recent years are tending toward distribution of data processing and storage facilities to the geographic location of the origin and/or end use of data. From the perspective of end users, distribution of the data processing function accommodates the need for accessibility to data critical to their decision making responsibilities. This is likely to represent a major motivating factor on the part of management. Further, a move to a distributed database environment may realize real hardware/software cost savings to organizations. Such savings may represent an additional, or perhaps the only, motivating factor.

The CODASYL Systems Committee, recognizing the above factors and concerned with the prevailing perception that database technology is applicable and consistent only with the philosophy of centralization, has undertaken a study of the implications of database technology in a distributed processing environment.

WHAT IS A DISTRIBUTED DATABASE ENVIRONMENT

Considerable attention has been given recently to computer networks and to database technology, but little to the application of database technology in a network environment. The traditional single-site orientation of DBMS and databases must be extended to operate in the context of a network of computer facilities. To provide a basis for discussion and investigation, the committee first laid a careful foundation of terminology. The following paragraphs define network and database related aspects of a distributed processing environment.

The network provides the underlying configuration of computer systems and communication facilities within which data is stored, DBMS's operate, and users access data. A node in the network consists of computer processing facilities (ranging from a large multiprocessor computer to an intelligent terminal) and an associated operating system sufficient for executing user and DBMS processes (programs, queries, etc.). In addition, data and its definition may be stored at a node. The precise structure of a node is an architectural design choice independent of the manner in which the node is connected to other nodes and the extent of geographical separation.

A communications facility is the collection of processes and physical facilities which interconnect the nodes. The communications facility includes knowledge of the physical location of each node, the physical path connections between the nodes, and the protocols to be used in sending messages between nodes. Processes in the communications facility will accept a message from one node and deliver it to another node or broadcast it to some or all other nodes. Two nodes may be connected directly or indirectly through other nodes. A network access process (NAP) exists at every node as the interface between processes at the node and the communications facility. The NAP is that portion of the communications facility which executes on the processing facilities of a node.

A traditional single-site database environment (See Figure 1)[1,2] consists of a database, a DBMS, a database definition ("schema"), and a user schema.* Placing these data-related

* User schema broadly refers to the user's view of the database. CODASYL has called this a "subschema", but, in general, the only requirement is that a mapping exist between the user schema and the "schema." It is not strictly a subset of the schema.

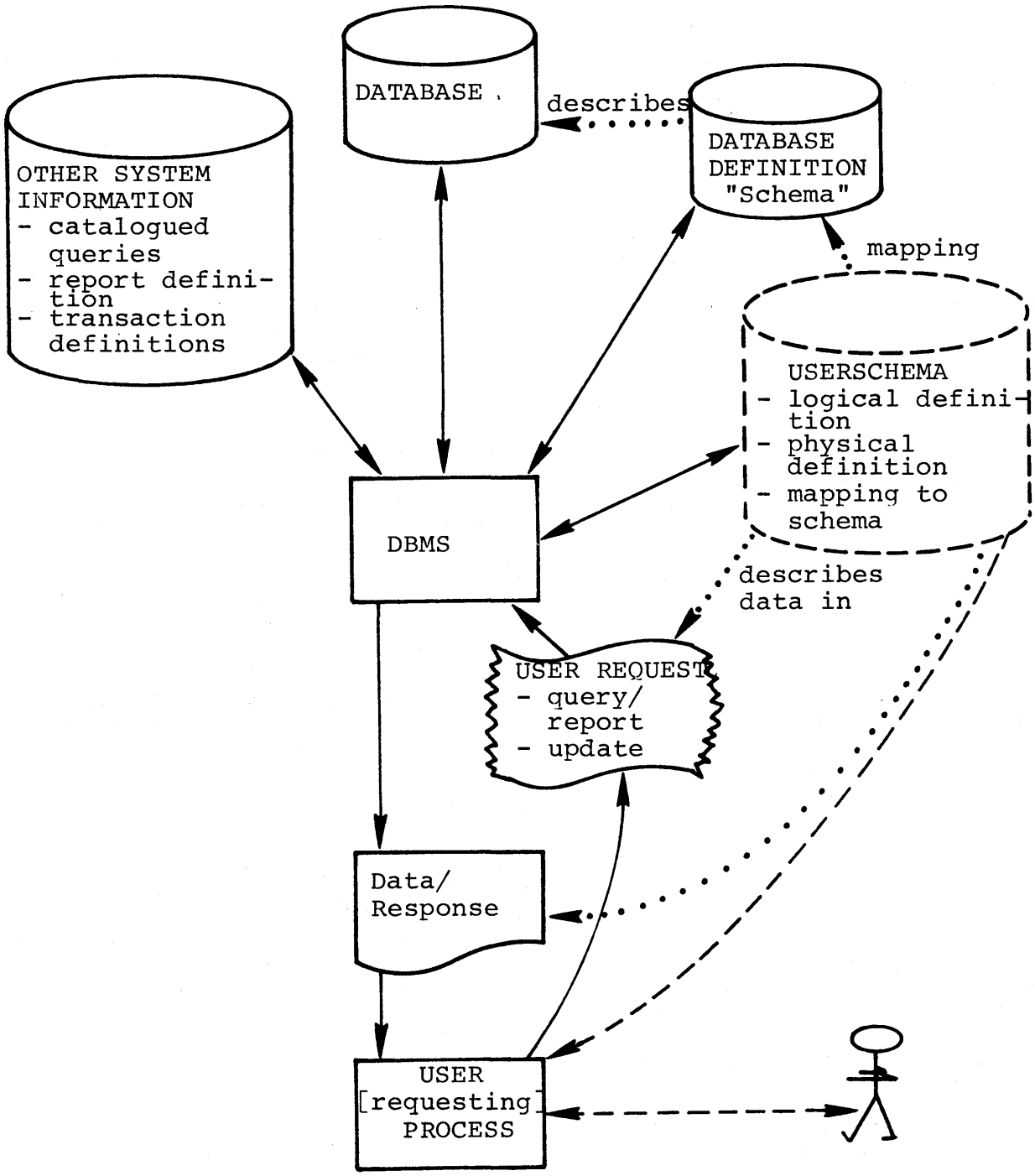


Figure 1—Conventional single-site database environment

components at the nodes in a network environment produces a distributed database environment. The primary architectural design choices relate to the distribution of databases, and the distribution of DBMS functions. All the databases may be stored at one central location (the traditional single site), they may be stored at different nodes, a given database may be broken down into pieces ("partitioned") with the pieces stored at different nodes, or finally, multiple copies of a database or its pieces may be stored at multiple nodes ("replicated"). Whenever a database (or part thereof) is stored, there also must exist appropriate data definition and DBMS capabilities to access and process the data. When the user request and the target database exist at different nodes, DBMS functions can also be distributed and performed at multiple nodes.

The integration of database technology and a network environment leads to new problems and the need for new functions. Perhaps the most important need is for some network-wide intelligence concerning the node location of all the databases in the system, their partitions and their replications. The network data directory serves this function.

When the system receives a user request to access data, a new DBMS function must first determine what data must be accessed and the node or nodes on which it resides. Then it must communicate with another DBMS. Increased heterogeneity within the distributed database environment, i.e., multiple different DBMS's, necessitate a translation of the request. Data conversion may also be necessary if the data is stored in different forms, according to different data structure models or on different hardware/storage devices. These are some of the key technical problems to be addressed by the data processing community in the coming years.

DESIGN ALTERNATIVES

Three levels of perspective

The Committee recognizes three distinct, but nevertheless interdependent, perspectives in providing design alternatives for a distributed database environment.

- Each component of the environment and the interrelationship of components. (See Figure 2)
- The study, in static state, of a node in the distributed environment including its various alternatives for configuration. (See Figure 3)
- The specification of user interaction in a distributed environment, emphasizing the sequence of events and the execution of functions, creating a dynamic model of the environment.

Components and interrelationships

There are five major components within a node of a distributed database environment as illustrated in Figure 2. Non-distributed databases have three of these—the database, the DBMS, and the user process. A distributed database environment requires two additional components—a network DBMS and a NAP. Figure 2 shows three types of nodes:

- A "complete" node with all five basic components.
- A "user" node with only a user process accessing data in the network.
- A "data" node with only those components necessary for data to be made available throughout the network.

Figure 3 provides a more detailed picture of "complete" node. To convert a conventional database environment (See Figure 1) to a distributed database environment requires the addition of the following components:

- Network DBMS
- Network Data Directory
- NAP (described above)

With a centralized database, which is the case in a single-site data processing environment, all the intelligence relating to the database is in one place. A conventional data definition can provide all the information needed for a DBMS to locate and process the stored data.

As soon as data becomes distributed—single, whole databases at different nodes, partitioned databases, or replicated databases—there must exist some information within the system which indicates where the different databases, their partitions, and their replications are stored. This is the primary role of the network data directory.

Network data directory

The network data directory contains information indicating the nodes at which the various units of data reside within the distributed processing environment. For example, given a file name, the network data directory provides the node or an indirect reference to the node(s) at which the file resides. It does not contain information about the physical location of nodes or the routing between nodes; this information is part of the communications facility.

Network database management system

If we assume that a DBMS only includes those functions which relate to a local database and that it has no cognizance of any other data nodes (i.e., a conventional, single-site DBMS), then all other new functions needed in a distributed environment can be packaged in a new system called the Network Database Management System (NDBMS). In the development of a distributed processing environment, it may be desirable to integrate the NDBMS functions with the DBMS, with the NAP, or leave as a separate package. We will be more concerned with the nature of the function and its information inputs, than with any particular packaging approach.

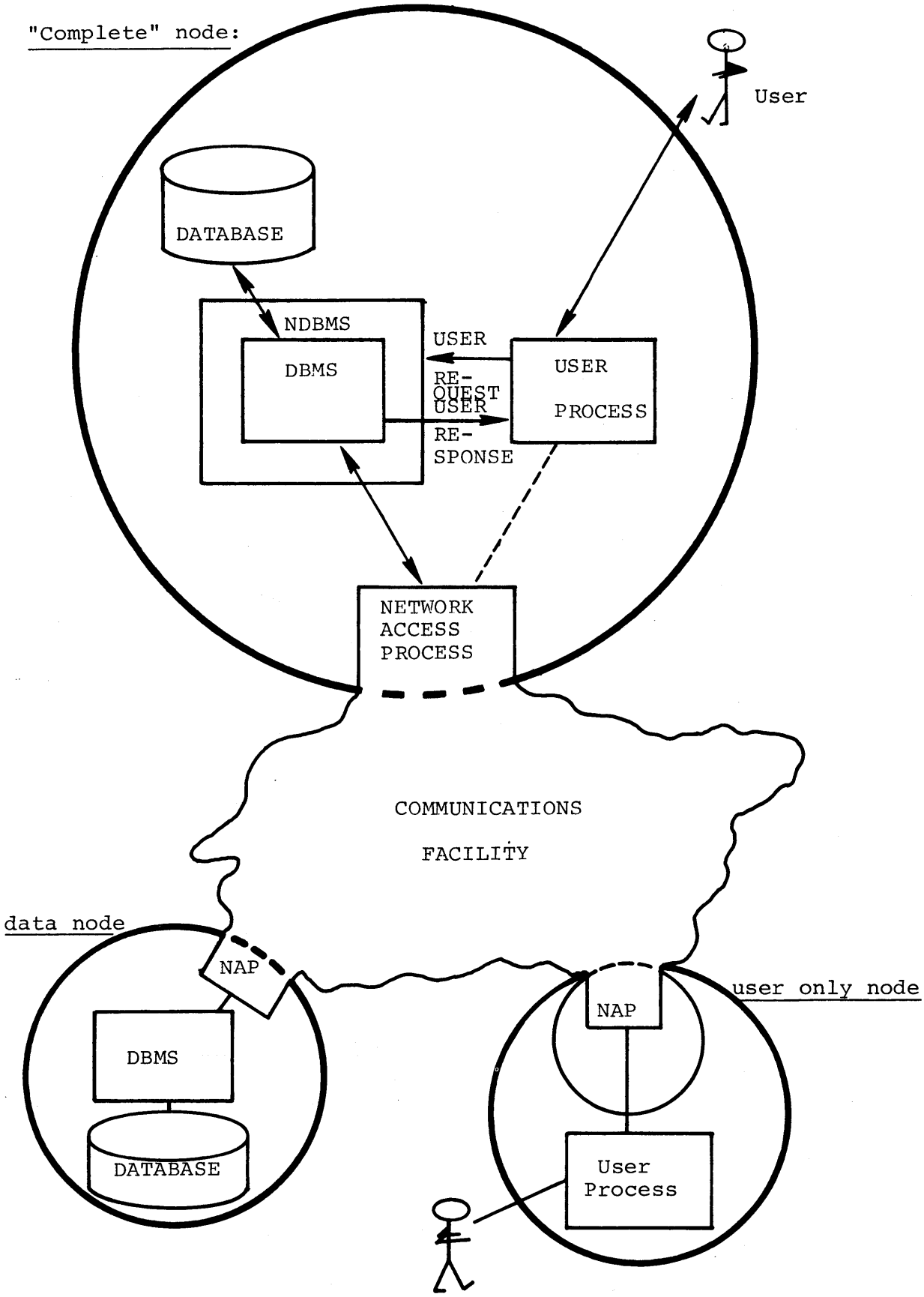


Figure 2—Data and users in a distributed database environment

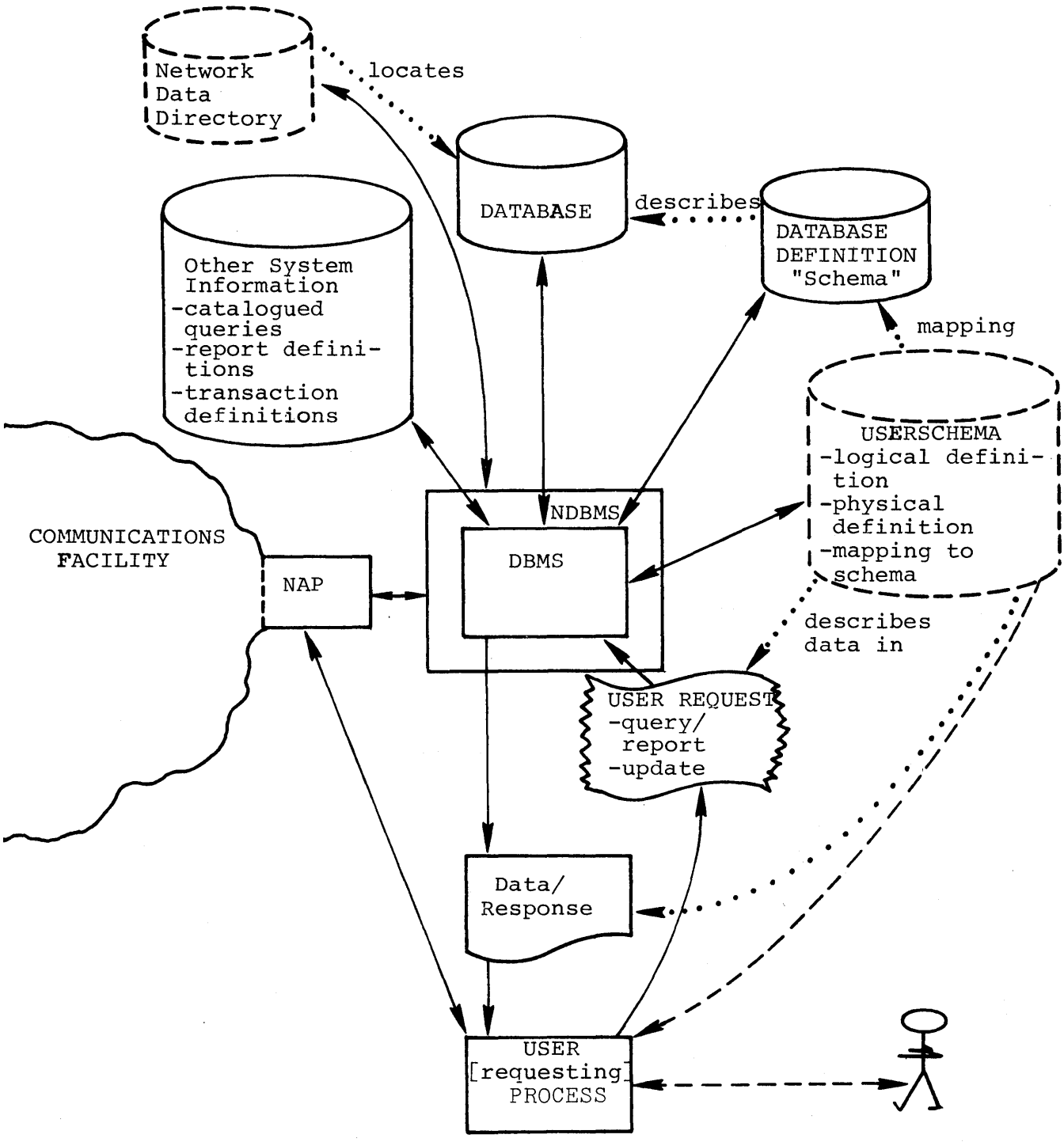


Figure 3—A "complete" node in a distributed database environment

The network DBMS includes at least the following functions:

- Intercept a user request and determine where to send it for processing, or what nodes must be accessed to satisfy the request.
- Access the network directory or at least know how to request and use the information in it.
- Coordinate the processing and response to a user request if it spans nodes, that is, the target data exists at multiple nodes.
- Function as the communication interface between the user process, the local DBMS, and DBMS's at other nodes (via the NAP).
- Provide data and process translation support in a heterogeneous distributed database environment.

Design choices and distribution alternatives

The specification of design choices and distribution alternatives for various distributed database architectures requires a model that will facilitate the study of database related distribution alternatives. If a major component does not exist as a single copy at a single node, then there are multiple alternative design choices as to its distribution to nodes.

- *Replication Versus Partitioning of Components*—the replications of a distributed component (or any piece of a component) are functionally identical copies of the component. Replication introduces the issue of update synchronization.

  The partitions of a distributed component are different and non-overlapping. Under partitioning, the whole of a component consists of multiple pieces which the system must be able to associate. The system must know the relationships among the pieces and be able to manage the whole component and its pieces.
- *Homogeneous Versus Heterogeneous Implementation*—a central issue relative to the replicated and partitioned distribution alternatives regards the relative degree of homogeneity/heterogeneity of the supportive components (namely the DBMS, schema, non-schema and system information) across nodes. If the corresponding supportive components at each node are different (heterogeneous) then the complexity and technological difficulty implementing the system increases. If all components are the same (homogeneous) then it is easier to implement and operate the environment. Given today's technology, the ability to design, implement and operate a distributed environment is greatly facilitated by "standardizing" components at each node. Thus, for example, utilizing the same DBMS uniformly at every node greatly simplifies the design of a distributed environment.

Figure 4 summarizes these design choices:

IMPLEMENTATION \ COMPONENT	Replicated	Partitioned
Homogeneous	The component exists in multiple identical implementations	The component exists only once in the system and has identical implementations
Heterogeneous	The component exists in multiple copies, but in different forms of implementation	The component exists only once in the system, but varies in its implementations

Figure 4—Alternatives for distribution of a component

These simple rules are complicated by observations of existing distributed environments.

- Hybrid distribution alternatives are possible as are time variant ones.
- Not all distribution alternatives are relevant to realistic systems.
- Not all combinations of component distributions are internally consistent or useful.
- Complex interdependencies exist.

TECHNOLOGICAL AND ADMINISTRATIVE ISSUES

Benefits from the distribution of data and its management are not without cost. Benefits resulting from the dispersal of data processing and DBMS capabilities must be balanced against the new technological and administrative problems which result in costs. The solution of these problems will demand a significant cost commitment in terms of additional hardware, software and communications facilities. Similarly, distribution will impact at the administrative level. Data control is complicated since the distributed environment and administrative control must be more highly coordinated. For the purposes of its report, the Committee distinguishes between technological and administrative implications.

The Committee has thus far in its deliberations identified a number of issues which sharply focus the technological implications of distribution. Each of these is briefly described.

Initial implementation problem

The first step in planning for a distributed environment involves careful engineering of the transition from existing operations to the new. Managing the conflicts of heterogeneity creates a significant problem. Often disparate database management systems may be in use, different processing hardware may be installed, and user languages may be incompatible. Conversion from the existing environment to a distributed environment is a technological issue.

Access control

A centralized DBMS facilitates the implementation of security restrictions. Through security mechanisms, protection is afforded against unauthorized, deliberate or inadvertent access to data in the system. One important mechanism used to achieve this goal is access control. In a distributed environment, access control is more difficult to implement because of the necessity of deciding where to locate the control mechanism. The vulnerability of dispersed data may be a real issue. In the extreme, it may be necessary to locate access control with each distributed component, a costly decision. On the other hand, centralizing access control may unduly burden the system with performance inefficiencies. However, if highly sensitive data is localized to specific nodes, higher assurances of strongest access control can be made.

Concurrency

A DBMS must include provisions for the control of concurrent multiple updates to a single record occurrence. In a distributed environment, the record occurrence itself may be partitioned across nodes. The control of concurrency can then become a more complex issue.

Synchronization

A DBMS takes into consideration the management of known redundancies such that a single update affecting multiple copies of the same record occurrence are effected in a manner transparent to the user and such that the database is always at the required level of consistency. The geographic dispersion of replicated copies makes this task more difficult.

Translation problems

In a distributed environment the number of conflicting logical views increases with the amount of heterogeneity in the network. The ability to effect the translation of data, processes, programs, user requests, and procedures from one node to another is a basic requirement of a distributed environment. The ability to implement transparent translation mechanisms presents a technological challenge.

Continuity of operation

An integral part of a DBMS's objective is the ability to operate continuously. This entails specific provisions for the recovery of data and the restart of operations. In a distributed environment this task becomes more difficult when applied to partitioned distribution, and when considered in conjunction with the synchronization issue.

Performance issues

The ultimate goal is to provide information services to the user. Success in measured, in part, in terms of delivery of timely results. The underlying rationale of distribution is availability of data. If the implemented system cannot perform efficiently enough to meet response requirements, the service goals are not met and the system fails. When distribution of components is achieved, there is a risk that the communications overhead incurred in the interest of system integration and control will degrade performance sufficiently to cause the user to question the responsiveness of the system as a whole. The balance between distribution and control will thus be a key performance issue.

Administrative issues

The design, implementation and operation of a distributed environment will be impacted by a number of administrative issues. The fact that these issues cannot be resolved at a technical level make them all the more important and deserving of attention.

Constraints on data flow

The distributed environment introduces the potential for increased amounts of data storage and flow at and across geographically dispersed locations. The assumption that data can be moved subject only to economic considerations is no longer valid. Constraints on the storage and flow of data may be expected from a number of sources, such as:

- *Legal*—governments will legislate constraints on data flow in the form of tariffs and regulatory law.
- *Privacy*—privacy requirements and restrictions will have an effect on the storage and flow of data.
- *International Law*—the flow of data across international boundaries is just beginning to be examined. Responsibility for control, authorization, nature of transmission, rights to impose legal restrictions, traffic constraints, etc., remain to be established.

Coordination and control of data

The administration of data resources is recognized as an important prerequisite for successful implementation of shared use of data. Distribution of data complicates this issue. The development and enforcement of standard practices is more difficult with dispersion, the coordination of common data descriptions is a problem when users are geographically dispersed, and the resolution of conflicting data needs becomes more tedious and time-consuming with distribution.

Organizational issues in administration of data

The emergence of an organizational entity (Data Administration) to address the foregoing issues has provided users with an administrative mechanism to deal with the problem.

In a centralized database implementation, the data administration function has often been organized to be controlled by a senior level data processing officer. In a distributed environment, however, data administration may be potentially exercised at each node in the network. Issues of organizational placement, interaction and coordination must be carefully scrutinized.

User acceptance

The distributed environment must, in the final analysis, serve the user. There are a number of phenomena which may cause concern in this regard.

- Resistance to change.
- Conversion from existing user procedures.
- Potential risk of complexity.
- Organizational realignment.

In a distributed environment each user at every node must find it attractive to participate or the system will fail.

CONCLUSION

The impact of "distributed processing" is already manifested in many sectors of the EDP community. The impact on database technology is significant; the subject of current deliberations of the CODASYL Systems Committee is forging a framework for dealing with these new architectural concepts and is establishing a basis for dealing with the topic on common ground and with clear understanding of the issues and considerations.

REFERENCES

1. Feature Analysis of Generalized Database Management Systems, CODASYL Systems Committee, May 1971. Available from ACM.
2. Selection & Acquisition of Database Management Systems, CODASYL Systems Committee, March 1976. Available from ACM.

BIBLIOGRAPHY

1. Blanc, R. P., "Annotated Bibliography of the Literature on Resource Sharing Computer Networks," NBS Special Publication 384, September 1973.
2. Blanc, R. P., "Review of Computer Networking Technology," NBS Tech Note 804, January 1974.
3. Bray, Olin H., "Distributed Database Design Considerations," NBS/IEEE Symposium on Computer Networks; Trends and Applications, November 1976.
4. Canning, R. G., "Distributed Data Systems," *EDP Analyzer*, Vol. 14, No. 6, June 1976.
5. Canning, R. G., "Network Structures for Distributed Systems," *EDP Analyzer*, Vol. 14, No. 7, July 1976.
6. Cotton, I. V., "Network Management Survey," NBS Tech Note 805, February 1974.
7. Fry, J. P., "Distributed Databases: A Summary of Research," The University of Michigan, Data Translation Working Paper, DE801, August 1975.
8. Irby, Hsing and Voss, "Transparent Intelligence Language Facility," Final Technical Report INCO, INC/1051-0676-TR-20-D(F), June 1976.
9. Kimbleton, S. R., "Computer Communication Networks: Approaches, Objectives and Performance Considerations," *Computer Surveys*, Vol. 7, No. 3, ACM, September 1975.
10. Kimbleton, S. R., "Network Operating Systems," *AFIPS Conference Proceedings*, Vol. 45, 1976.
11. Lowenthal, E. I., "A Survey—The Application of Database Management Computers in Distributed Systems," 3rd VLDB Tokyo, October 1977.
12. Lowenthal, E. I., "The Backend (Database) Computer, Part I & II," Auerbach Information Management Series; Database Management, Pennsauken, NJ, Auerbach Publishers, 1976.
13. Maryanski, F. J., "A Survey of Developments in Distributed Database Management Systems," *IEEE Computer*, February 1978.
14. Rothnie, J. B. and N. Goodman, "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.
15. Thomas, Robert H., "A Solution to the Update Problem for Multiple Copy Data Bases which Uses Distributed Control," Bolt, Beranek, and Newman, Report #3340, (for ARPA), July 1976.

Commentary on the CODASYL systems committee's interim report on distributed database technology

by CHARLES BACHMAN

Honeywell Information Systems
Waltham, Massachusetts

The author has reviewed the Interim Report on Distributed Database Technology and is responding from the vantage point of the ANSI/X3/SPARC/Study Group on Distributed Systems. This study group is attempting to establish an architectural approach which will put all aspects of distributed systems into perspective, including distributed databases. The author has been studying this problem as part of Honeywell Information System's investigation for the last three years.

My first comments must be addressed to the overall document which is well written and conveys its message well. I am in agreement with everything in it and will address myself to those aspects which it does not cover at the present time. As an interim report, one cannot criticize it severely for having some gaps. It has been circulated for comment and is soliciting comments so that the final report may be more accurate and more complete. I recommend the reading of this report to all interested parties and urge each reader to comment so that the final report will be as complete as time permits.

The interim report lists a number of motivations for distributed systems. All are valid in one situation or another. There are a couple of additional reasons which I would like to add to the list. The first reason is a human factor reason. This is to permit some element of the business to physically possess the part of the corporate database which holds its part of the data. This gives them the responsibility and the satisfaction of updating their database without removing it logically from the required access by others. Another human factor reason relates to the distribution of the database to reduce its vulnerability to strike action or acts of terrorism. Another reason is the matter of database size vs. computer power. There is no computer on earth which can by itself handle all the data processing for a large corporation. Therefore, processing must be distributed and is already distributed for most of the FORTUNE 500 companies. In most cases they are already physically distributed but not logically integrated and will not be logically integratable until "distributed database facilities" become available.

There should be one item added to the list of reasons for the centralization of data processing. This is to achieve a concentration of expertise sufficient to run a large data processing operation. From an administrative point of view, this is very important. Learning how to distribute the expertise along with distributing the processing and database may be a real roadblock.

From the vantage point of the ANSI Study Group on Distributed Systems, the report seems to lack a certain architectural overview which would put distributed database management into perspective. The ANSI distributed systems reference model begins with the notion that the information processing aspects begins with the notion that the information processing aspects of an enterprise should be viewed as a network of cooperating workstations. Each such workstation is a "location" where a certain kind of work takes place. This workstation may be manual, industrial or computerized. This workstation supports one or more processes which execute the procedures of the workstation and which primarily access the locally stored data. Access to remoted stored data is acceptable, but known to be slower and more expensive. The processes of the workstations cooperate by exchanging messages which request certain actions and transfer such data as appropriate.

Where the process and the data it accesses are in the same machine, data access operates essentially as it did before it was distributed. Where the process and the data are in separate machines, more elaborate mechanisms are necessary. Those described in the CODASYL interim report are representative. One thing which appears missing from the report is any discussion of how processes and data would be partitioned and set up in the various machines in the network. While this is a packaging issue, my current guess is that it is a critical one. Probably, as critical to distributed database access as the "clustering" concept is to the performance of our current centralized database. Simply speaking, clustering is the placing together in the same page as many records of a set occurrence as possible. Where one set has been chosen in favor over another because of the

anticipated frequency of access. Frequency of access is also the heart of the database vs. process distribution questions. It has been said that there are two approaches: (1) take the data to the process and (2) take the process to the data. The carefully preplanned distribution of processes and the data which they are most likely to access to the same machine is the most important step which avoids much of the processing burden associated with distributed processing.

Within the ANSI reference model, three alternate schemas have been discussed to achieve access to distributed databases. From our point of view, the picture presented by the CODASYL interim report describes the most difficult to design and implement. It will also be the least efficient under certain operating circumstances. The three approaches discussed are characterized as;

(1) cooperating user workstations
(2) cooperating storage management workstations
(3) cooperating database management workstations (CODASYL)

All of these approaches assume that a facility for process to process exchange of information is a basic capability of a distributed system. The difference in the three approaches is who is talking to whom and what is the nature of the information exchanged.

The cooperating user workstation approach assumes that the users are knowledgeable of the data file vs. process (workstation) configuration. Thus, if a process cannot get what it needs from the local database, it will formulate a message to be sent to another workstation which is co-located with the data which needs to be processed. When the message arrives at the workstation, a process of that workstation will access the message and carry out whatever local access and updating is necessary and respond with an answering message. When workstations are considered as elements over which the total work load has been divided, then it does not make much difference if one is designed to approve a credit request for a new sales order or whether one is designed to answer questions about the Minneapolis inventory. It is only a slightly further move to design a workstation which receives data manipulation commands as a message text, executes these commands in terms of its local database management system, and sends a response in accordance with its success.

The cooperating storage management workstation approach is based upon the fact that most of our database systems are built as virtual memory systems and do "page turning" independent of the operating system and hardware. When a page is requested by the database management subsystem the storage management subsystem asks whether the page is in a main memory buffer or whether it must be fetched from the disc? If it must come from the disc, then the question is which disc drive and address? With a distributed storage management system, the third question must be asked, my disc or one managed by some other storage management subsystem? If it belongs to someone else, then a message must be formulated with the request. A response message will either return the page, state the process must be waited until the page is available, or stating that a deadlock would result if the process were to be waited. The cooperating database workstation is essentially that one described in the CODASYL Interim Report.

The CODASYL report discusses several kinds of control tables: network directory, concurrency control, integrity control and security control. There is a discussion appearing several places in the report which questions whether these tables should be centralized or distributed. From my point of view, there is only one answer: distributed. The centralized approach has all the problems which the distributed database system is attempting to side-step. e.g., the vulnerability of the entire system to the failure of one element, the congestion problem, etc.). What must be done is to give each node the information about its local resources, process and only information about remote objects of local interest.

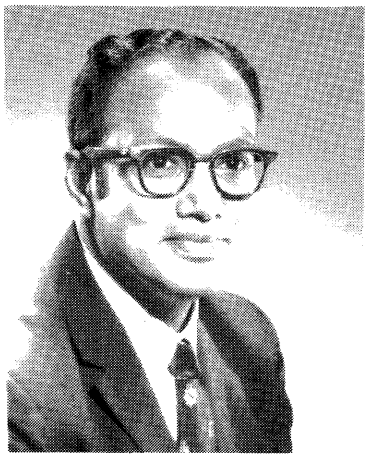
One note about the concurrency control is relevant, the problem in a distributed system takes on complexity beyond just the aspect of multi-nodes. Most concurrency control systems have a secondary responsibility to be sure that two or more processes are not waiting for each other in order to access data (deadlock). With the introduction of message exchange between processes, it is possible for process (1) to be waiting for a message from process (2) which is waiting for a record held by process (1) . . . deadlock! It also must be considered that a process may be waiting upon a terminal operator action. If that terminal operator could possibly be waiting on the action of another process in the system, then that anomie must also be included in the path of waiting processes to determine if deadlock exists.

The question was raised concerning the effect upon concurrency if a record were to be split between nodes. This isn't a problem because the assignment and wait logic can work at any level of granularity. It could work at the level of the logical elements: item, item group, segment, record or file. It could work at the level of physical elements: page, group of pages, track, cylinder, or extent. As long as the control is maintained non-ambiguously, it will work. The only difference is the smaller element requires more bookkeeping and the larger elements force processes to wait and possibly be recovered from deadlock when the effective access desired would have caused no interference.

The text described two basically different forms of distribution: distribution of parts of the only copy and distribution of copies of the whole database. Where updating is a pertinent issue, the distribution of copies seems less efficient than distributing parts of the only copy and getting it updated at its only location. We have seen several large systems in the field which have chosen a variation which is worth describing. It could be considered as a form of the multiple copy approach. In this case, major portions of the file are copied and partitioned to be sent to a number of transaction processing machines. These transaction machines handle the on-line daytime processing of their partition of the database. All on-line transactions are also sent to a centralized system where they are batched with additional off-line transactions and reprocessed in a nightly batch run. At the end

of the nightly run, the updated centralized database is again copied and updated partitions are sent to each transaction machine.

So much for the extension of the generalities. The real question is where do we go from here. I see the only satisfactory long term answer in the form of a set of international standards which permit all kinds of computers, software, communication facilities and terminals to be joined together to form an extended network of cooperating workstations. I say extended because the workstations served will need to communicate within companies and between companies and across international borders. No one vendor or one country is large enough, smart enough, or influential enough to go it alone. This is the reason I have supported the ANSI effort and the ISO effort to move toward this standard. We have set ourselves an impossible schedule to reach the goal of international standards. The end of 1978 is the goal for an internationally accepted reference module (levels, protocols and interfaces). The end of 1980 is the goal for the first new protocol and interface standards which will support the interprocess exchange of information. Working groups are being established to study and prepare specifications. One of these is concerned with the distributed database access and control. I expect that the CODASYL System Committee with its members and report will make a significant contribution to making those standards. Are you ready to move and move rapidly? Are you ready to help write and negotiate these standards?



Area Co-Director:
C. V. Ramamoorthy
University of California at Berkeley
Berkeley, California

Area Co-Director:
Gordon Bell
Digital Equipment Corp.
Maynard, Massachusetts

Computer architecture

The recent advances in Computer Technology, providing unlimited possibilities for an ever expanding growth of the application spectrum, have created substantial research activities. The technical sessions in the architecture area of this conference are conceived and planned to expose the profound advances in hardware technology and also to reveal their significant impact on the current and future systems architecture. This series of sessions presents the state of the art and also some of the thought provoking challenges to the designers of the future systems.

The session the "Impact of Semiconductor Technology on Computer Architecture" traces its impact of semiconductor technology on the computer architecture and the emergence of microprocessor systems. It also highlights a discussion on how the future data processing technology would be influenced by the semiconductor technology and the associated problems. The evolution of this technology and future projections are discussed by the leaders of this emerging technology.

The sessions on "Evolution of Architecture" trace the evolution of architectures of well known computer families. These sessions articulate the major challenges and accomplishments of these systems. They also discuss the lessons that these systems have taught and how the future systems would be built. These two sessions would define certain cautious guidelines for the future systems.

In the future the user is expected to be his own architect. By specifying his requirements the future user would be able to configure his system from the off the shelf subsystems. In order to do this certain tools and techniques are needed by the user. The session on "User Impact on Architectures" explores such issues related to user involvement.

In order to define the directions for future large scale architectures it is essential to look back and assess the effectiveness of existing large scale systems. Session on "Large Scale Computer Architectures" looks at the evolution of large scale systems and discusses how effectively these systems have satisfied the user needs.

The dramatic reduction in hardware costs due to the LSI technology make it economically feasible to expand the application spectrum and to cover the areas which are hither to unexplored by developing special purpose architectures.

Session on "Special Purpose Computer Architectures" highlights architectures for one of the important application areas—pattern recognition.

In spite of tremendous advances in computer technology the existing intracomputer standards are in chaos and some are confusing. With the result there exists a large degree of incompatability between the subsystems of various manufacturers thus imposing several artificial constraints on the designers who use these subsystems. The session on intra-computer standards will extend a discussion on the impact of standards on computer architecture. Organizations involved in developing these standards will discuss their current thoughts and focus on how uniform standards could be evolved that would benefit both the user as well as the computer architect.

Intra-computer standards

by TSE-YUN FENG
Wayne State University
Detroit, Michigan

The need of standards on many industrial products is quite easy to understand. Ideally, we would like to have various parts of a product interchangeable. From a user's point of view standards provide him with desirable convenience and reductions in product cost as well as costs for maintenance and repair. In some respect, even manufacturers stand to gain from standardization. Yet, the progress in standardization on various products has been slow. This is particularly true in the case of computer standards.

Since the introduction of the first electronic digital computer some thirty years ago the standards activities on computers have been relatively small. There are several factors that may contribute to this phenomenon. One is that until recently computer systems have been quite bulky involving many components and circuits that are proprietary or patented. The number of major companies manufacturing computers are relatively few and each of them has its own system of standard equipment. Also, the maturity of a product is usually a prerequisite for standardization, but the rapid advances in computer technology prevent the standardization in many areas. It is thus not surprising to find that much of the recent national standards effort has been mainly in the language area since they are relatively machine independent. Languages such as FORTRAN, APT, COBOL, PL/1, BASIC have been standardized by the American National Standards Institute (ANSI). In addition, there are also activities on character sets, and information processing vocabulary. Recently, IEEE Computer Society took the initiative in proposing and developing software standards. These projects are on Software Development, Software Quality Assurance, and Software Documentation.

On the hardware side, the standards activities have also been limited to those that are not closely related to the central components of computers. They are mostly on magnetic tapes, disc packs/cartridges, codes, character recognitions, and interface circuits. Recent advances in integrated circuits make it possible to have a large number of memory bits and their associated circuits implemented on one chip. A number of these chips can then be interconnected to form a larger memory. Furthermore, such large scale integration also permits designs for a small processor to be implemented on a chip. The microprocessor chips together with memory modules can be used to form larger computer systems. The impact of the integrated-circuit chips on future computer architecture is quite evident. Several standards projects on semiconductor memories and microprocessors have also been initiated by IEEE Computer Society with the participation of both manufacturers and users. On semiconductor memories, the projects involve Electrical Description of Semiconductor Memory Devices, Test Pattern for Testing Semiconductor Memories, and Thermal Properties of Memories and Other IC's. In addition, a number of new standards projects relating to semiconductor memories are being initiated. In the microprocessor area three new standards projects, Floating Point Format and Algorithms, Assembly Language Syntax and Lexicography, and Relocatable Object Module Formats are in progress.

It should be noted that there are also parallel efforts on computer standards by the federal government, the military, as well as Joint Electron Device Engineering Council (JEDEC), the International Purdue Workshop, etc.

Software standards—With hints of their relation to computer architecture

by H. HECHT
Consultant
Los Angeles, California

INTRODUCTION

Software standards can help the developer, the user, and regulatory agencies in facilitating communications, in extending software life, in broadening the applicability of programs, and in reducing the personnel and organizational reorientation required in transitioning from one project to another.

Communications are facilitated by use of standardized terminology, by use of standards for development methodology (e.g., structured programming), and by standardized test reporting to name just a few pertinent areas. Facilitating communications implies not only that misunderstandings are avoided but also that the software procurement cycle can be shortened and procurement costs reduced because both contracting parties may from prior experience know their obligations when a standard is invoked. Use of a standard also reduces the cost of software development because the allowance for coping with unfamiliar specifications can be reduced.

Software standards can extend the useful life of programs by facilitating adaptation to new versions of operating systems, by making it possible to transfer the program to a new computer, and by making maintenance and updating of the program itself a lot easier and cheaper. The broadening of a program's applicability is made possible by the features cited above plus uniformity among users of program format, test, and test reporting requirements that can be accomplished by standards.

Much time is required during the start-up phase of critical programs in order to train otherwise experienced analysts and programmers in specific contractually imposed design, coding, or documentation requirements. Allowances have to be made for errors in complying with unfamiliar requirements, and at times management may be reluctant to bid on a software development that is within its normal range of business interests simply because of uncertainty regarding a test or documentation requirement that had not previously been encountered. Software standards could reduce this wasted effort in transitioning from one project to another.

In view of these benefits one must ask why the few existing standards are not more widely employed and why there is not more effort at implementing software standards. Part of the answer lies in the newness of software engineering as distinct from programming for one's own use. The latter implies the translation of a defined sequence of operations into machine-readable form, an activity that was best managed in a local context and to which standards are about as applicable as they are to the writing of business letters. Software engineering recognizes that requirements analysis, test, validation, program maintenance and related tasks, in addition to programming proper, are necessary to provide the computing service that the user expects and needs. This broader activity involves the interaction of many groups, extends over a long period of time, and is in scope quite like a construction project. Hence, standards are appropriate to guide the activities.

But together with the recognition of the benefit of standards must also come the recognition that they involve costs. A standard for documentation may require a more detailed format than is essential for a given application, and a software quality assurance standard may impose access control that will be perceived as burdensome by personnel that arranged the mounting of their own test tapes as they saw fit. In the context of computer architecture it is appropriate to point out that just about every software standardization activity will increase memory and throughput requirements. For example, standards for test or test documentation may invoke analysis tools that slow down the execution of the primary program. So right at the outset let it be said that whatever can be done to increase the speed of computers and the size of the easily accessible memory will be welcome for the application of software standards.

In terms of scope, software standards can be divided into three broad categories: language standards, software engineering standards, and software implementation standards. Each of these classes will be separately discussed below. There is another important classification of standards by issuing agency: voluntary industry standards, semi-official standards (e.g., those promulgated by international agencies), federal standards, and military standards. This paper will be principally concerned with non-governmental standards.

Finally, the common use of the term standards encompasses some documents that are actually recommended practices, guidelines, or specifications. These documents

obviously have a lesser enforceability or more restricted applicability than true standards, but, in view of the rather limited effect of any software standards on computer architecture, these distinctions are not too important in the present context.

LANGUAGE STANDARDS

Standards promulgated by professional organizations are presently applicable only to High-Order Languages (HOLs).[1-8] As such, their effect on computer architecture is indirect; clever compiler designers, if necessary, can cope with almost any machine instruction set. Military agencies may be motivated to standardize the assembly level instructions for certain classes of computers, primarily those that process tactical software. Implementation of such standards would have a much more direct effect on architecture, but discussion of this event is beyond the scope of this paper which emphasizes voluntary compliance.

We should also be careful to point out that the effect HOLs have on computer architecture is to a large measure independent of the existence of language standards. Thus, the adoption of features such as general register organization and floating-point processors that are a boon to processing FORTRAN object code preceded the adoption of the ANSI FORTRAN standards. Standardization of HOLs may nevertheless be significant in this connection because it increases the portability of code and makes the benefit of architectural features that support HOLs more visible to the user. Since source code written in a standardized HOL can be executed on any computer for which a compiler written to that standard exists, it can become quite apparent to a user interested in such matters how both compile time and run time are affected by the computer organization. Hence, standardization of HOLs in general is a positive factor in promoting architectural features that support processing of HOL-derived code.

So far the discussion has dwelled on use of compilers for translating a program coded in a standard HOL for generating the instructions that ultimately are operated on by the computer. Another approach for handling HOL source code is through direct-execution machines. Considerable research in this field has been under way for some time but so far leading to full-scale development of only a few direct-execution computers.[9] Among many factors responsible for the slowness of this development may have been the proliferation of language "dialects", requiring translation or other special processing before existing source code could be processed on a direct-execution machine. HOL standardization may remove at least this factor among the obstacles to acceptance of architectures for direct execution.

Block-oriented languages, such as ALGOL, benefit particularly from provision of stack features in the computer. Further standardization of block-oriented languages may therefore become a factor in promoting stack architectures.[10]

The general tendency toward increased use of standardized HOLs (likely to be heightened by a recent DoD directive) will tend to benefit those computers that can efficiently process code derived from them. However, it must also be realized that going from assembly code to HOLs will require more memory and greater processing speed if the same computational capabilities are to be provided. Thus, the previously mentioned effect that standardization demands faster computers with larger memories is evident in the area of language standards.

SOFTWARE ENGINEERING STANDARDS

In the broader sense, dictionaries for information processing that have been available for some time[11] could be considered software engineering standards. Also, some Department of Defense acquisition management documents contain material related to software engineering standards.[12] But specific software engineering standards are only in the draft stage, e.g., a Military Standard for Tactical Software Development was circulated as a draft in June 1977. It is also interesting to note that the Federal Information Processing Index provides for software engineering subjects, e.g., operating procedures and operating systems, but that there are at present no entries under these headings.

Effectiveness of software procurement and development suffers, and in some cases the introduction of computers for critical tasks is being hampered, by this lack of standards. The IEEE Computer Society Subcommittee on Software Engineering Standards is now active in preparing three standards in this area: software development, software quality assurance, and software test documentation. A specialized terminology for software engineering has also been drafted by this committee.

The software development standard is expected to require a modular organization, some form of structure programming, and "readable code." These requirements do not easily translate into effects on machine architecture but they will in general result in an expansion of code over that generated without them and hence again will demand greater memory capacity and speed.

Standards to be drafted for the quality assurance area may, among other things, be concerned with controlling access to software and with recording of all executions of a program during test or certain other critical phases. In current practice compliance with such requirements can be monitored by making use of features of advanced operating systems. As a result of the standardization the requirements for such controls may become sufficiently common that it may be profitable to explore architectural features that can simplify the demands placed on the operating system. In the quality assurance area there is an existing Military Specification on Software Quality Assurance Programs that has no apparent impact on computer architecture.[13]

Test documentation standards will be concerned with areas such as reporting completeness of test, complete description of the test environment, and accurate recording of all changes to software or the environment over the usually rather extended period required for test of critical software programs. One of the methods for reporting completeness of test, but probably not the only one that would be acceptable under

such a standard, is the recording of all node exits taken during a series of test runs. In current practice this can be done by use of staic analyzers to locate the nodes, and subsequent "instrumentation" of the nodes such that passage during execution can be counted. Execution time of such an instrumented program can be orders of magnitude longer than that of the original code, and the need to insert and then to remove the instrumentation may impair validation of the test. This is one of the major obstacles to more widespread use of these software test tools, and architectural features could possibly be brought to bear to simplify instrumentation requirements and to reduce the execution time penalties.

STANDARDS FOR SOFTWARE DELIVERABLES

Software can be delivered to the user in a large number of ways: installed in a specified computer, as a listing, as card decks, as paper tape, and as magnetic tape to a number of formats. Within most of these there are further variations on character set, density, and other format variables. Physical dimensions of the deliverables and format features are, in many cases, covered by standards or trade practices. In most cases these affect the design of computer peripherals much more than computer architecture proper. As a matter of fact, there seems to be a tendency, through the use of microprocessors, to make peripherals "smarter" (i.e., to adapt themselves to format variations) and thus to remove the need for some of the conversion tasks that must now be handled by processing in the central computer.

Software documentation is an important deliverable that is covered by a number of voluntary standards.[14–15] In addition, this area is the subject of numerous Department of Defense specification and data item descriptions (DIDs). None of these documentation standards appears to have an immediate or even remote impact on computer architecture.

CONCLUSIONS

The imposition of software standards will, in several areas, require longer code (i.e., more storage) and slower execution (i.e., require a faster computer in order to handle a given task). Thus, software standardization imposes a requirement for more powerful computers. It would, however, be probably more correct to say that the capability of industry to provide more powerful computers permits standardization rather than the other way around.

The requirements for HOL processing, in particular the requirements of newer HOLs, may favor certain architectural features in the central processing unit, and the fact that these languages are standardized may impose a further urgency on the computer community to adopt architectural features that support these languages.

Software engineering standards may become accepted in the near future. These will require some code expansion, control of software access during critical test phases, and may lead to more widespread usage of automated test tools to quantify completeness of test. Requirements likely to be formalized by these standards are currently being met by software but hardware features to support the software or to supplant it may at some future time be worth investigating.

Standards for software deliverables primarily affect peripherals. Some unburdening of the central processing unit by "smart" peripherals may be in the offing. Standardization of output formats may further simplify the central processing requirements in servicing peripherals.

REFERENCES

1. Basic FORTRAN ANSI X3.10—1966
2. ANS FORTRAN ANSI X3.9—1976 (also, ANSI X3.9—1966)
3. Programming Language APT ANSI X3.37—1974
4. Programming Language COBOL ANSI X3.23—1974
5. Programming Language PL/I ANSI X3.53—1976
6. Minimal BASIC ANSI X3.60—1976
7. Programming Language Minimal COBOL ANSI X3.65—1977
8. Programming Language IEEE/AIRINC Std ATLAS Test Language, IEEE Std 416—1976
9. ACM/IEEE Symposium on High-Level Language Computer Architecture, University of Maryland, 7-8 November 1973, IEEE Cat, 73-CH0811-0-C.
10. *Computer* issue on Stack Machines, May 1977
11. American National Dictionary for Information Processing X3/TR-1-77 (based on the ANS Vocabulary for Information Processing, ANSI X3.12—1970)
12. Management of Computer Resources in Systems, Air Force Regulation 800-14.
13. Software Quality Assurance Program Requirements, MIL-S-52779(AD).
14. Software Summary for Describing Computer Programs and Automated Data Systems, Federal Information Processing Standard (FIPS) Publication 30
15. Guidelines for Documentation of Computer Programs and Automated Data Systems, FIPS PUB 38 (1976)

Standards for semiconductor memory

by J. REESE BROWN, JR.
Burroughs Corporation
Piscataway, New Jersey

INTRODUCTION

The concept of standardization and the advantages which it offered by-passed the semiconductor memory industry for the first five years of its existence. Standardization has never been easy to achieve in the modern electronics industry, but it has been doubly difficult for the fledgling semiconductor memory industry.

This difficulty can probably be attributed to the highly competitive nature of the industry combined with the high technology levels involved in the products. Each company had what they felt were advanced proprietary concepts, processes, and designs. This, they felt, would give them a competitive edge so they were reluctant to talk to the competition regarding standards, for fear that they might lose some of their commercial advantage.

The lack of standardization extended over a broad range of aspects of the industry. The language used by each company in describing its product was different from that used by its competitors, even when the products were intended to be interchangeable.

Beyond the lack of communications standards, there was the more fundamental problem that the product images and characteristics were not standardized. At one time there were over four different 1K dynamic MOS RAMS. The situation got worse with the advent of the 4K dynamic RAM. At least seven different noninterchangeable designs were announced, each described in a different language.

The first 4K devices to be introduced were in a 22 pin package. The twelve address bits were supplied in parallel. From a performance standpoint, the parts were generally compatible. However, the two companies which led the field chose to use different pinouts, making the parts incompatible. In addition, a third company introduced a part which was compatible with one of these except that the output was different. It had a low level constant current rather than a TLL compatible voltage out.

Following the original 22 pin devices, there were three different 18 and 16 pin devices introduced. The two 18 pin devices used different pin-outs and interface logic. The 16 pin design used a two clock, multiplexed address interface logic.

In addition to the gross differences of pin-out and logic, there were numerous cases of subtle differences in timing requirements and interface signal levels which created incompatibilities between parts.

Another aspect of Semiconductor Memory where we have been hampered by the lack of standards is related to device test. A critical aspect of testing is the test pattern used. This is the address and data sequence which when combined with the power supply voltages and interface voltages and timing, are used to determine if a device meets specifications or not. Often, in marginal devices, minor variations in the test pattern can make the difference between detecting a bad part or not. It is essential that we be able to communicate the exact details of these test patterns so that tests may be duplicated by vendors and customers.

Several years ago, a group of engineers representing both vendors and users recognized the problems which resulted from this lack of standards and set off to correct the situation. They formed a committee and then petitioned the IEEE Computer Standards Committee to sponsor their work. The group became formally recognized as the "Semiconductor Memory Standards" sub-committee. The membership is made up of representatives from device manufacturers, manufacturers of computers and related equipment, and memory test equipment manufacturers.

The group normally meets twice a year at the solid state circuits conference in February and at the semiconductor memory test symposium in October.

The sub-committee is made up of several task groups which meet more frequently to prepare drafts of standards to present to the sub-committee. At the present time there are three task groups working on standards. The first of these is working on standards related to the electrical description of Semiconductor Memory devices. The second group has addressed the problem of describing the test pattern used to test Semiconductor Memory devices. The third one, which is now being organized, will develop technical standards for describing and measuring the thermal properties of memories and other integrated circuits.

STANDARDS IN PROCESS

There are three standards currently in various stages of preparation. The first one is being developed by the device description task group: It is entitled "Proposed Standard for

Semiconductor Memory Data Sheet Generation". The intent of this standard is to provide a common language and format for the industry to use in describing the electrical and mechanical properties of Semiconductor Memory devices.

The document is divided into four sections. The first section title "Product Description" lists a series of different items of a general nature which should be included in the specification of any product. It also gives some definitions and logic conventions which are recommended to be used so that the data sheets and specifications can easily be understood by everyone.

Section two, "Product Specification" gives recommendations for the actual electrical and mechanical specifications of the device. In the various sections, ground rules, definitions, and conventions to be used are given. For example, it contains a statement which gives a common definition of the basis for generating the "Absolute Maximum Ratings" section. In the past, there was never a definition of what this means. Every company had its own ground rules for generating these numbers, or in some cases, no ground rules. As a result, the numbers are often arbitrarily chosen and therefore meaningless.

In other areas, definitions and conventions are given to standardize symbology and the presentation of the data. The section on timing specifications is one which has suffered from the largest differences from vendor to vendor and as a result, the greatest amount of confusion amongst the users. The timing specification of current dynamic MOS memory devices is extremely complex. There are typically seven different signals or groups of signals, each of which have up to four critical timing events. Each of these events must be specified with respect to at least one other signal and often with respect to three or four others. The result is a timing specification which often contains a table of over 30 parameters with either a minimum or a maximum value, and sometimes both. For this standard, the task group has invented a symbology and procedure for generating timing parameters to relate any timing event to any other timing event. The result is a uniform, nonambiguous "language" for the timing specification. A summary of the new symbol procedure is included in Appendix A.

In a similar, though less complex way, the voltage and current portions of the specification are covered. Preferred symbols are given for the different lines as a function of their use within the device.

The third section covers the area of use related information. It includes recommendations for the inclusion of information which will simplify the users task of applying the device. The items covered are: graphs showing the interaction between operating variables, current waveforms, bit maps, equivalent interface circuits, testing information, and application information.

The standard is concluded with a series of appendices which give: (1) a summary of input/output pin names, their symbolism and definitions, (2) a format and example of timing specification for a part using the conventions of the standard, and (3) a summary of the timing parameter symbol procedure.

The second standard being worked on is one relating to the description of test patterns for RAM Memory. The original intent of the task group was to accomplish two tasks. The first was to develop a language which could be used to uniquely define the address and data sequence used in a memory test pattern. The second was to use the results of the first task to stabilize the definition of some of the more commonly used patterns such as march, galpat and walking 1/0.

One of the original ground rules for the language development was that it should be machine independent and not rely on the readers knowledge of the programming language or architecture of any of the currently popular memory testers. The reason for this is the wide disparity between tester pattern generators used. Some work with an absolute address scheme and others use relative addressing. Some define data as a simple sequential string of bits, while others define data as a spacial function of the addresses, while at least one has the capability of doing both.

In its initial efforts, the task group attempted to apply one of the more widely understood high level programming languages, such as APL or Basic. As the standard went through successive drafts, the language evolved into one which tends to be closer to the language of some of the testers. It is undergoing further refinements to satisfy the objections of some of the more hardware oriented sub-committee members.

When the language is stabilized, the task group will then start to develop a repertoire of test patterns which are in common use in the industry. This is being done, not to give support to any patterns as more effective than others, but to improve communications so that when someone uses a pattern name, everyone knows exactly what he means.

The third standard is one which is concerned with the thermal resistance of an integrated circuit. In the past, the terms, symbols, and definitions for I.C. thermal resistance have not been standardized at all. Each vendor and user tends to have his own private set. In addition, some of the definitions used by device manufacturers have no real value to the user in the environment in which he uses the parts.

The work has progressed to the point of having a first draft which has been circulated to the task group for discussion. It will be at least late 1978 before it has been refined to the point where it is ready for presentation to the subcommittee.

The draft standard starts with a series of thermal resistance definitions. The list includes the traditional parameters $R_{\theta jc}$ and $R_{\theta ja}$, which are favored by the device manufacturers. It also contains several new ones which allow the user to predict or measure the I.C. junction temperatures under system operating conditions.

The definitions are followed by a section on test methods for measuring thermal resistance. It includes discussions on test fixtures, test devices, test instruments and procedures.

The end result of this effort will be an all inclusive standard which serves the needs of device designers, manufacturers, and users. At the present time, there is no applicable standard for anyone, only a variety of sometimes conflicting traditions.

The work of the sub-committee will continue until those

topics which are the proper province of the IEEE have been considered.

OTHER STANDARDS WORK

There is another class of standards which is badly needed by the industry but which is not properly handled by an organization like the IEEE. This is the standardization of the interface image and specifications of the memory devices themselves. It was this lack of standards which allowed four different 1K devices and seven different 4K devices to be introduced into the market. There is, however, a committee which has been formed, to address this problem. This committee was formed by JEDEC (Joint Electron Devices Engineering Council) as committee JC-42. This committee is made up of representatives of device manufacturers only, however, there is close communciation between the IEEE Semiconductor Memory Standards sub-committee and JC-42 so the users voice is well represented.

In addition to the work of JC-42, there are some additional standards work being pursued by other groups. The JEDEC committee JC-11 is preparing standards on the mechanical packages used to house memory devices. Some of the smaller "chip carrier" packages which are now being considered by this committee can have a long range effect on memory architecture. This can come about as a result of the increased packaging density which the use of these packages will allow. In addition, their use opens up the possibility of cost effective multi-chip packages with higher levels of complexity.

There are numerous standards being generated by the N. B. S. or through the MIL specification channels which relate to the Semiconductor Memory field. They are broad based standards which are intended to cover the general field of intergrated circuits. One of these is worth noting since it covers topics being considered in the IEEE thermal resistance standard. This work, done by G.E., has been published in the report RADC-TR-77-321, "Thermal Resistance of Microelectronic Packages."

This is an excellent piece of work which covers in a broad way definitions and test methods relating to package thermal resistance. The work of the IEEE committee is an expansion of this work to make it more broadly useful to the commerical users of integrated circuits.

CONCLUSION

The Semiconductor Memory industry has suffered for its lack of standards. The result has been a needless waste of resources, both by the vendors as well as the users. The progress of the industry has been retarded in ways which it is impossible to quantify. The task of making up this lack has now been undertaken by two separate groups: (1) an IEEE Committee working on standards related to communication and definitions, and (2) A JEDEC Committee working to standardize product definitions and images. The combined results of these committees will greatly assist the Semiconductor Memory industry to develop new, better products and the users to apply them more effectively.

APPENDIX A

SYMBOLS AND ABBREVIATIONS

This data sheet uses a new type of specification nomenclature that is derived from background work of several users and manufacturers of semiconductor memories. It should help to clarify signal and parameter symbols and definitions and make data sheet information more consistent.

ELECTRICAL PARAMETER ABBREVIATIONS

All abbreviations use upper case letters with no subscripts. The initial symbol is one of these four characters:

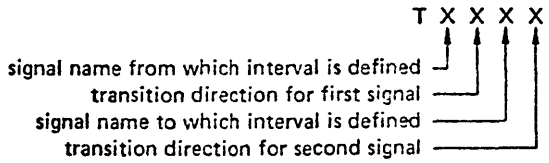
V (Voltage)
I (Current)
P (Power)
C (Capacitance)

The second letter specifies input (I) or output (O), and the third letter indicates the high (H), low (L) or off (Z) state of the pin during measurement. Examples:

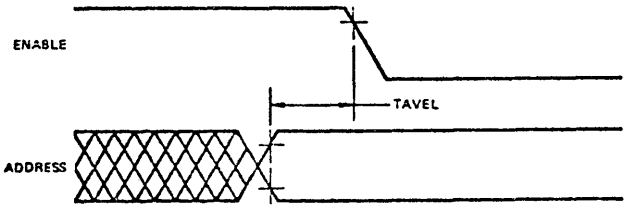
VOH = Output high voltage
IIL = Input low current
IOZ = Output off current (leakage)

TIMING PARAMETER ABBREVIATIONS

All timing abbreviations use upper case characters with no subscripts. The initial character is always T and is followed by four descriptors. These characters specify two signal points arranged in a 'from-to' sequence that define a timing interval. The two descriptors for each signal point specify the signal name and the signal transitions. Thus the format is:



Example:



The drawing shows the address setup time defined as TAVEL, Address Valid to Enable Low time.

The signal definitions used in this data sheet are:

A = Address
D = Data In
Q = Data Out
W = Write Enable
E = Chip Enable

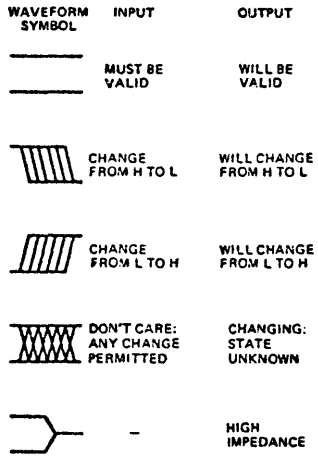
The transition definitions used in this data sheet are:

H = transition to high
L = transition to low
V = transition to valid
X = transition to invalid or don't care
Z = transition to off (high impedance)

TIMING LIMITS

The table of timing values shows either a minimum or a maximum limit for each parameter. Input requirements are specified from the external system point of view. Thus, address setup time is shown as a minimum since the system must supply at least that much time (even though most devices do not require it). On the other hand, responses from the memory are specified from the device point of view. Thus, the access time is shown as a maximum since the device never provides data later than that time.

WAVEFORMS



Microprocessor standards

by TOM PITTMAN
Itty Bitty Computers and Homebrew Computer Club
San Jose, California

and

ROBERT G. STEWART
Stewart Research Enterprises
Los Altos, California

Now that microprocessors have been with us for three or four years, we can stand back and observe some disadvantages of our free enterprise system, as well as obviously great advantages. The disadvantages tend to fall most heavily on the shoulders of the user, not the manufacturer. The proliferation of microprocessors with incompatible instruction sets, adds unnecessary chores to learning new mnemonics, and requires rewriting (at large expense) existing software. Another interesting situation can be seen in the case of the 8080 and the Z80 processors whose mnemonics differ for precisely the same instructions. The act of copyrighting mnemonics by microprocessor manufacturers places the problems in the user's laps in order to protect the proprietary interests of the manufacturer of the chip.

Whenever standards are proposed, two reactions are voiced, polarizing the opinions. On the one hand we hear general disdain: "Who needs them?" or "Standards impede progress." On the other hand we hear cheers: "What took so long?" or "Lack of standards impedes progress." The field of microprocessors is not immune to this division.

We will have standards in software as well as in hardware. Unlike the larger mainframe computers, where the entry costs are high enough to insure a virtual monopoly on software and interface standards by a single manufacturer, there are a half dozen or more significant contributors to microprocessor software for each chip type, and as yet none of their products are compatible with each other. There are hundreds of components designed for very few processors, and many of these cannot even coexist in the same system. Standards are inevitable because the users will not tolerate this chaos very much longer.

The absence of standards for the large, batch computers today is simply due to the fact that the manufacturers fight standardization. Plus compatible peripherals are a touchy example. It is not in the mainframe companies interest to foster a standard interface. This reluctance to standardize extends to software aspects which directly affect a user such as job control streams, editor commands, disk I/O calls, tape formats, internal data representation, ASCII vs. EBCDIC, and pairing of source and relocatable files. These differences are particularly vexing for someone who must use computer systems made by a number of different companies. If there were only one computer company, then standardization efforts would not be necessary. The interesting thing is that the present situation looks like it results from a condition where all the different computer companies believe that they are the only one! Note that most of the problem areas which would benefit from standardization would have relatively little or no impact on the computer architecture!

The fun aspects of the hobby or homebrew computer run headlong into the software and hardware idiosyncracies of the different manufacturers. For example, because the Port designations for the console and the console status flag's ready bits have not been standardized, the act of bringing

"We can't do business after all... my computer isn't speaking to your computer."

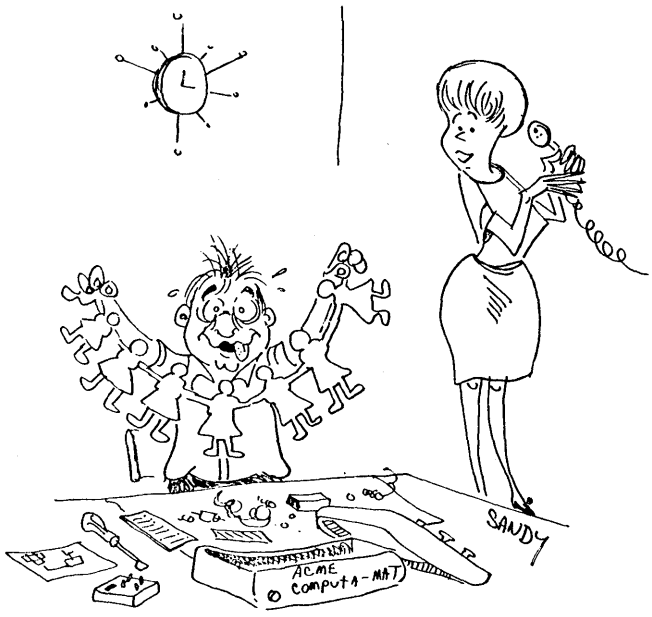
up a system is rendered much harder since a friend with a working system may not be able to substitute his proven software without first patching in the required changes. This patching can be quite difficult because the hardware may be flaky also.

Note that these types of problems are seldom serious to the manufacturer, but represent hurdles to the novice user that make him curse the people responsible for making life difficult for him. And typically those people are cloistered within companies and make arbitrary decisions without regard to the problems that result for users interfacing with hardware and software from a number of sources.

A veritable jungle has come into existence in the homebrew or personal computer hobby. Memory boards and DMA devices for the S-100 bus are an example. The manufacturers state that they are S-100 compatible, and indeed they are unto themselves. However memory boards from different manufacturers are, in some cases, not compatible with each other. The initial MITS description of the Altair or S-100 Bus did not nail down the timing of the bus. By this absence, the timing defined by Intel for the 8080 processor must intrinsically be recognized as the original timing standard for that bus. Later, other processors such as the Z-80 and the 2650, have been used on the bus, without having the same timing signals and relationship as the 8080, so that memory and DMA devices are not directly interchangeable.

Usually it is the user who takes the lumps in both cost and frustration because the manufacturer had stated the boards are bus compatible, when they actually were not because of their timing and control characteristics.

The IEEE Computer Society Microprocessor Standards Committee was formed in the summer of 1977 to try and resolve some of the obvious problems that have arisen in the use and applications of these marvelous chips called microprocessors. The veritable complete revolution in technique which the microprocessor has brought to the electrical engineering profession will be aided for years to come by a codification attempt at this time.



"I've got the Acme people on the line. They say they are sorry but your kit contained instructions for a KI-8 kit and you bought a KP-2 kit. They hope you have not been inconvenienced."

SOFTWARE STANDARDIZATION EFFORTS

The IEEE Computer Society has taken a bold step forward in forming a standards committee to deal with the most pressing of these issues. We have selected three specific areas of concern in the software area where immediate action can be of tremendous benefit in the microprocessor community:

1. *Floating point format and algorithms*—We expect to be soliciting comments on a first draft of a proposed standard by the time of NCC. In this standard we will be defining forms for both normal floating point operations and for extended range and/or precision. Overflow and underflow conditions will be specified, and minimal requirements of the algorithms implementing these operations will be defined.
2. *Assembly language syntax and lexicography*—There are major questions subject to compromise in this area. It is expected that this standard will define the assembler syntax for each of the more popular microprocessors, specifying both mnemonics and operand coding. By considering all of the processors at one time it is hoped that the similarities between processors will be reflected in the mnemonics and operand syntax, and that guidelines may be developed for the assignment of mnemonics to new microprocessors as they appear. In particular, it is hoped that assemblers supported by timesharing services will all accept the same source code for the same chips (this is not now that case), and that their reduced development costs in bringing up additional processors will make wider support for new CPUs more attractive.
3. *Relocatable object module formats*—This area, though of vital importance, seems to be moving the slowest. Again, the major thrust of our efforts is the specification of an adequate object module format for each of the major processor families.

Here also it is expected that the careful selection of functional and lexical characteristics will insure a uniformity across processor lines which will therefore also extend into future CPU design. Clearly the burdens placed on the relocating linker are somewhat arbitrary, so part of our work includes the analysis of the kinds and extent of the tasks which should be deferred to that phase of program translation. Obvious candidates for support are the output of macro assemblers (see standards effort #2 above) and of FORTRAN compilers. The requirements of other high level language compilers should also be considered, as well as the possibility of a proper subset (with reduced functions) for minimal operating systems.

One of the frequent objections we hear points to the dissimilarities between different microprocessors, with particular reference to the one-chip controllers. Since most of these are single-sourced and lack the multiple-manufacturer proliferation of software development tools seen in the 8-bit processors like the 8080 and the 6800, we have not been overly concerned about extending compatibility in these directions. However, most of the committee members have had extensive experience with several microprocessors, and our emphasis in each of three areas is focused on the similarities rather than the differences between the varieties of chips under consideration.

HARDWARE STANDARDIZATION EFFORTS

The initial hardware problems tackled by the Microprocessor Standards Committee are standardization of several of the existing and proposed busses, namely the S-100 bus, the Intel MDS Bus, National's Microbus, and the PI bus proposed by Pietsch and Khalsa. After the bus issues have crystallized, the Committee will look at problems in the Small Business and Hobby Computer area, such as interfacing and protocols, standards for memory peripherals, data formats, disk formats, etc.

The key issues that arise in standardizing the de facto busses are timing diagrams, DMA control, defining unused pins for future address, data space, and control extensions, and accommodating the higher clock rates of future processors. The proposed standards for the busses will be given at NCC and the audience will have the opportunity to make comments on them. They will also be able to make suggestions for future standards activity by the committee.

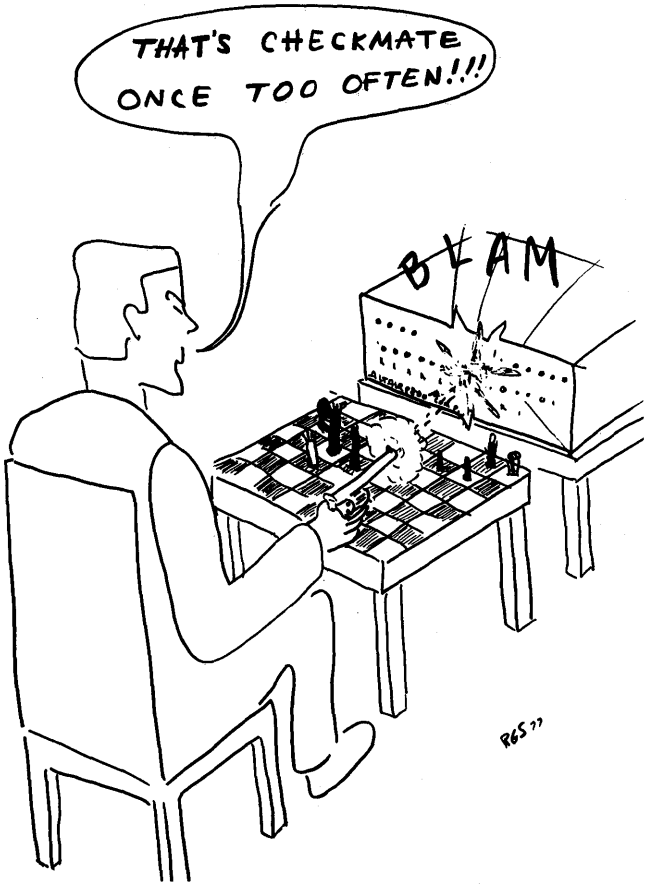
Of the existing de facto busses for microprocessors, the most widely used is the bus MITS used on its Altair 8800 computer announced by Ed Roberts in Popular Electronics magazine in February, 1975. (Many computer hobbyists regard that as the world's first digital computer.) The wide variety of plug-in boards that are available for the Altair of S-100, at competitive prices, have enabled the bus to serve the hobbyist and the small business market reasonably satisfactorily. The problems that have arisen with the S-100 bus are: (1) The use of positive true rather than negative true logic, (2) an overabundance of control and non-essential signals on the bus, (3) the use of separate data in and data out busses, (4) the use of microprocessors with different timing characteristics than the 8080, (5) different voltage power lines directly adjacent to each other, (6) one common AC and DC ground, and to some extent, (7) poorly designed boards which plug into the bus.

The efforts of the IEEE Computer Society Microprocessor Standards Committee to resolve some of the S-100 bus problems have been spearheaded by Howard Fullmer and George Morrow. The timing and DMA protocols adopted define processor-memory interactions in a manner which is more tolerant of non-8080 processors. All timing relationships are specified only with respect to the phase two ($\phi 2$) clock signal. Any device proclaiming itself to be a bus master, such as a DMA device as well as a processor must generate a set of 18 control signals as well as the address and data signals. Some DMA devices have failed to generate the full set of bus control signals and so have led to incompatibility problems with different memory boards which were relying on the presence of those signals. The 8224 system controller can be used by DMA devices and non-8080 processors to generate the control signals needed on the S-100 Bus. The adoption of standards for the bus will enhance its utility in the years to come and help preserve the investment which tens of thousands of hobbyists have made in their equipment.

The capability of the chip manufacturers to develop ROM chips to store specialized software also will act to force standards on higher level languages as they relate to microprocessors, development systems, operating systems, and the personal computing market. Such chips will provide BASIC, FORTRAN, PASCAL etc. interpreters and compilers at the selection of the user. Thus the advance in hardware technology acts to force upon us the standardization of relocatable code formats discussed earlier.

Packaging standardization for microprocessors has evolved to the predominant use of a 40 pin package. As longer word length processors are created, more pins may well be required unless extensive bus multiplexing is used. There is some tendency now to use pins on opposite diagonals for ground (pin 20) and power supply (pin 40); however, even that very limited pin-out standardization is not at all universal. Chip designers lay out the chip with all the logic capability they can, and only worry about the package pin-out at the end. If a pin-out standard is imposed for microprocessors, then the chip designer would have to consider the pin-out in the initial chip design, and may have to run traces for longer lengths than otherwise in order to reach the pads. This acts to reduce the available chip area for the device's logic functions. Thus there are significant semiconductor chip layout and utilization reasons to deter the formulation of a pin-out standard of μP packages.

A far more significant aspect of microprocessor architecture and design which would benefit from standardization right now is the "timing architecture," or precise sequence of signals that interface with external busses and support chips. In the practical sense of being able to use existing support chips and busses for a new or different microprocessor, the timing architecture is as (or more) important a conceptual issue as the register structure and logic capability. The dominant existing de facto busses have intrinsically been designed around the timing convention of the 8080. A timing standard for handshaking and control signals of microprocessors would be very useful in furthering the universality of busses, support chips, DMA devices, and controllers. Such a standard should have to be processor independent if at all possible. The work which our committee has undertaken on the S-100 bus timing is essentially committed to the same goals, and will probably be helpful in developing a standard timing architecture for microprocessors themselves. The committee has attempted to choose specific tasks which do not impact architecture in the usual sense, and the microprocessor manufacturers in Silicon Valley agree that our tasks would not.



The efforts by U.S. governmental agencies to impose standards on computers has resulted in the selection of a subset of existing architectures as being acceptable for future DOD procurement. Further, the instruction set of the AN/UYK 20 minicomputer is being established as the standard required instruction set for all future minicomputers. In addition, some governmental agency staffs active in standardization efforts believe that a standard bit pattern for the processor word is achievable for a given instruction. If such a requirement is imposed in the future, then the specific processor design and architecture will certainly be impacted. An instruction set standard can be dealt with by an assembler or cross-assembler; imposing a standard bit pattern in the word would require hardware or a chip design that would implement the standard. Whether the conventions that have been established by DOD for computers and minicomputers are forced downward on microprocessors remains to be seen.

The rapid changes in technology have led to multiple sources of software and hardware to create a system, as compared to a single source, bundled situation which has prevailed in the past. For the user to benefit from the remarkable advances in technological capability, it really is essential to formulate suitable software and hardware standards now.

ACKNOWLEDGMENTS

The cartoons by "Sandy" are copyrighted by Creative Computing Press and are used with permission. The authors have benefited greatly from conversations with other committee members, and the ideas and attitudes expressed herein are shared by many of them.

Reflections in a pool of processors— An experience report on C.mmp/Hydra*

by WILLIAM A. WULF and SAMUEL P. HARBISON
Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

This paper is a frankly subjective reflection upon the successes and failures in a large research project—the construction of a multiprocessor computer, C.mmp, and its operating system, Hydra—by those most intimately involved in its design, construction, and use.

C.mmp and Hydra have now reached a sufficient level of maturity to establish themselves as useful and reliable computing resources at Carnegie-Mellon University. The user community has grown from primarily operating system implementors to include researchers in other operating systems and multiprocessors and casual or curious users interested in using the unique features of the system (e.g., the Algol 68 language, whose first implementation at CMU was on C.mmp.).

Some of the scientific results we originally hoped for have been published and are listed in the bibliography at the end of the paper. Other results will be published in the future as we observe the system under varied loads and over longer periods of time. In addition to these factual results, however, we have learned a number of things of a more subjective nature—things that we did right and, perhaps more importantly, things that we did wrong. We believe that many of these lessons are not unique to our project, and their presentation here will be valuable to the larger computer science community.

For those people unfamiliar with C.mmp and Hydra, we shall provide a brief overview of multiprocessor research at CMU, and some details about C.mmp, Hydra, and the goals we originally set for the research project. This information should serve as a general background against which our evaluation of the project can be cast. The interested reader will find more details in the bibliography.

Multiprocessor research at CMU

In late 1971 we at CMU decided to embark on a research program to explore multicomputer structures—especially those structures in which the several computers share a common address space. At the time it appeared to us that the economics of LSI technology would make multi-mini or multi-micro structures the architecture of choice for many medium- to large-scale applications. In addition to the economic arguments, there appeared to be many other advantages to such structures, including high availability, expansability, and so on.

Despite the fact that a number of multiprocessor computers had been built prior to 1971, relatively little of a scientific nature was known about them. Our goal was to explore a number of alternative multiprocessor designs, examining both the hardware and software issues, and to report on these explorations. To that end we undertook the design and construction of two multiprocessor systems, C.mmp and Cm*, and their associated software.

C.mmp, the subject of this paper, is a relatively straightforward multiprocessor. Begun in 1972, it connects 16 processors to a large shared memory (up to 32 megabytes) through a central crosspoint switch. The access time from any processor to any word of memory is identical. Cm*, started in 1975, replaces the crosspoint switch with a distributed, bus-oriented interconnection scheme between processor-memory pairs. In contrast to C.mmp, the access time from a Cm* processor to a word of memory can vary by an order of magnitude depending upon the particular processor and memory module involved. These two machines have quite different implications on the software which runs on them; between them we are able to explore many of the interesting issues of distributed processing.

C.mmp

C.mmp is a multiprocessor composed of 16 PDP-11's, 16 independent memory banks, a crosspoint switch which permits any processor to access any memory, and a typical complement of I/O equipment. A path through the switch is independently established for each memory request and up to 16 paths may exist simultaneously. An independent bus, the IP-bus, carries control signals from one processor to another; no data is carried by this bus. Collectively the 16 processors execute about 6 million instructions per second;

* The research described here was supported by the Defense Advanced Research Projects Agency (Contract: F44620-73-C-0074, monitored by the Air Force Office of Scientific Research). The views expressed are those of the authors.

the total memory bandwidth is about 500 million bits per second. In short, despite the fact that it is built from minicomputers, C.mmp is a large-scale machine.

The current configuration of C.mmp includes 5 PDP-11/20 processors (5 usec/instruction), 11 PDP-11/40 processors (2.5 usec/instruction), and 3 megabytes of shared memory (650 nsec core and 300 nsec semiconductor). All of the 11/40 processors have been modified to include writable microstores; thus we are able to tailor their instruction sets to specific applications. The cost of this configuration is roughly $600,000, of which $300,000 is the cost of processors, $200,000 is memory, and $100,000 is the switch, IP-bus and other special equipment. Of course, there is an additional cost associated with I/O devices.

Hydra

Hydra is the "kernel" of the operating system for C.mmp; it is not intended to provide most of the familiar features of an operating system (e.g., it does not provide files, a command language, or even a scheduler). Rather, Hydra provides an environment in which it is (intended to be) easy to write user-level programs that supply these familiar facilities. Hydra was designed in this kernal fashion in order to permit (and encourage) experimentation with features and policies appropriate to multiprocessors.

Hydra, which was a research project in its own right, uses a capability-based protection structure, a scheme in which only the possession of the appropriate kind of reference to an object (e.g., a file) grants access to that object. In order to allow user-level definition of operating system facilities, Hydra extends the basic capability scheme with the ability to define new types of objects and (protected) operations on these object types. Thus it is possible for a user to define new types of files, processes, message buffers, or whatever. These newly defined types share an equal status with those that already exist—which is another way of saying that Hydra attempts to preempt as few decisions as possible, thus allowing the users to tailor the system to their needs.

Software already built on top of Hydra in this manner includes file systems, directory systems, schedulers, and language processors (for Algol 68, L*, and a flexible command language).

Project goals

Two general goals influenced both the hardware and the software design from the outset. The C.mmp/Hydra system was envisioned as both *symmetric* and *general purpose*. By *symmetric* we mean that replicated components, such as processors, are treated as an anonymous pool; no one of them is special in any sense. By *general purpose* we simply mean that we did not intend to cater to *only* those programs which need a multiprocessor; the multiprocessor character of the machine is used to improve throughput across a set of independent jobs as well as to multiprocess single jobs. Both the hardware and software were designed with these goals in mind.

The *symmetry* goal is manifest in a number of ways. At the hardware level, for example, an interprocessor interrupt mechanism was designed so that every processor could interrupt every other processor (including itself) with equal ease. At the software level there is no "master-slave" relation among the processors—any processor may execute any part of the operating system at any time (subject, of course, to mutual exclusion in accessing shared data structures). At the user level, a job may execute on any processor, and indeed may switch from one processor to another many times during its execution.

The impact of the *general purpose* assumption is more subtle; it implies that we have to provide a broader range of software than would be expected if our focus had been more narrow. It also implies that optimizations to a specialized problem domain should not be made in the operating system. Some of the specific effects of this goal will be found later in the evaluations.

Performance evaluation tools

Many of our evaluations of C.mmp are based on data obtained from a number of tools designed to measure system performance. Although not one of our greatest successes, we think these tools are important enough to present here. We have three measurement tools: a script driver, a hardware monitor, and a kernel tracer.

The Script Driver is a program which can place a measured load on the system by simulating a number of users at terminals performing various tasks. This known load can make the interpretation of performance measurements much easier.

The Hardware Monitor is a device built at CMU which can monitor in real time the signals on a PDP-11's bus. The Monitor is very useful in measuring the activity of a single C.mmp processor, and for recording the activity of small portions of the operating system. It is less effective in measuring total system performance.

The Kernel Tracer, the most commonly used tool, is built into the Hydra kernel. It allows selected operating system events (e.g., blocking on semaphores, context swaps) to be recorded while applications are running. The accumulated data can be processed off-line to give a detailed record of what was happening on each processor. Naturally, the use of the tracer slows down the entire system, but this obvious point doesn't really seem to matter in practice.

The importance of these tools should not be underestimated. In any system as complex as an operating system, design decisions are often based on intuitive assumptions of performance tradeoffs. Without accurate measurements, these design assumptions cannot be verified. Certainly we found that some of our assumptions were wrong, causing us to redesign several parts of Hydra.

Format of the paper

The body of this paper is a highly edited report of a meeting called specifically to evaluate the C.mmp/Hydra

project. The attendees were representatives of the various groups involved in the design, implementation, and use of C.mmp and Hydra; hardware designers, operating system implementors, those doing performance evaluation, and four major users. In all, sixteen persons attended, the maximum number we felt could interact productively.

The purpose of the meeting was to solicit the opinions of the participants concerning the nature of our successes and failures. We had also solicited written opinions from a wider group—in fact, just about everyone who has had anything to do with C.mmp and Hydra. The participants knew, of course, that the results would be reported in this paper.

The meeting and written responses produced over a hundred distinct comments. To organize these in a coherent fashion we asked the participants to decide upon our five greatest successes and five greatest failures. With some exceptions the comments have been organized under these headings; the participants' comments have been indented to separate them from background information and summary comments.

Any paper that sets out to reflect upon the successes and failures of a research project is potentially self-serving. We were extremely conscious of that danger and have attempted, through the format of the meeting and the editing of its transcript, to construct the paper in a manner which minimizes this effect. Either our initial fear of being self-serving was groundless, or the format chosen worked extremely well. We shall let the readers judge for themselves, but we feel that the result has been a reasonably objective, well-balanced view of the C.mmp/Hydra project.

OUR GREATEST SUCCESSES AND FAILURES

We shall begin this report with what, in fact, happened last at the meeting—a listing of our most notable accomplishments and mistakes. This list was created after all opinions had been expressed, thus the participants had the opportunity to hear the opinions of the others before deciding upon the content of the list. To keep the discussion crisp we arbitrarily chose to limit each list to five items. Surprisingly (to the editors at least), despite the differing interests of the participants there was essentially complete agreement on the items to be included on each list.

Our notable accomplishments:

- We constructed a cost-effective, symmetric multiprocessor.
- We provided, in Hydra, a capability-based protection system which allows the construction of operating system facilities as normal user programs.
- We were able to distribute the Hydra kernel symmetrically over all processors.
- We provided successful mechanisms for the detection of, and recovery from, software and hardware errors.
- We used an effective methodology for constructing the Hydra kernel.

Our notable disappointments:

- The hardware is less reliable than we would like.
- The small address of the PDP-11 has a large negative impact on program structure and performance.
- We are unable to partition C.mmp into disjoint systems.
- We did not put enough human-engineering into the software interface to the user.
- We did not give enough attention to project management.

Neither our successes nor failures are, of course, unqualified, and the story behind each is littered with smaller successes and mistakes. Moreover, there are dependencies between the things that went well and those that didn't; the fact that we have a running 16-processor system must be tempered, for example, by a poorer-than-expected reliability record. The reliability record, on the other hand, led us to greater concern for software structures that detect and survive hardware malfunction—and we count those structures among our most important accomplishments. For all these reasons, while we have used the success/failure list to organize the paper, one should not expect all the points listed under a "success" to be positive in nature. On the contrary, we believe it important to expose the contributing events, both positive and negative, as well as the major points listed here.

With that introduction then, here is the report of the meeting.

THE SUCCESSES

A cost-effective multiprocessor

C.mmp's design goals included speed, simplicity, and the use of as many commercially-available components as possible. Because C.mmp is a unique computer some critical parts had to be designed and built especially for the project. While this was a burden, it did give us maximum freedom in the design of these critical components, including the crosspoint switch, the IP-bus, and the processor modifications for memory relocation. These were all built by the CMU Computer Science Department Engineering Laboratory.

- The basic design goals have been justified by experience, with speed having been the least important emphasis.
- CMU-built hardware is not a large proportion of the total system cost.
- The crosspoint switch is very reliable, and fast enough.
- The use of immediately available components was a major factor in getting C.mmp built as fast as we did, but it limited us in taking advantage of technology which developed in succeeding years.

We were especially happy about the evaluation of the crosspoint switch, which many people thought would be C.mmp's Achilles' heel. In retrospect we think we were too concerned about raw speed in the design of the switch and

memory; as it turns out, most applications are sped up by decomposing their algorithms to use the multiprocessor structure, not by executing on a processor with short memory access times.

The comments at the meeting did reflect some specific complaints about the hardware, several of which we later decided were significant enough to be listed as some of our major disappointments. Many of these stemmed from our choice of a processor for C.mmp. In 1971, only the PDP-11/20 minicomputer met our requirements. In 1974 we decided to take advantage of technology advances and use the new, faster PDP-11/40 processors to complete C.mmp. One feature of the PDP-11 architecture which might be expected to impact the goal of symmetry for C.mmp is the close association of an I/O device with exactly one processor.

The PDP-11 processors required more modifications than we expected to ensure the security of the operating system.

The PDP-11's 16-bit address is too small for many interesting applications.

Having to supporting two PDP-11 models complicated the development of the processor modifications and the operating system. It would have been better to have had a single processor model, regardless of its speed.

Having I/O devices bound to particular processors made it difficult to move a device from a malfunctioning processor to a good one, but device utilization was not otherwise sacrificed.

Perhaps more than anything else, our experience with the PDP-11 has given us a much clearer idea about what features are really important in choosing a processor, and which are not. Our consensus is that speed is not very important, for reasons already cited in conjunction with the crosspoint switch. Reliability is very important, but we found that much can be done in software to increase the overall system reliability, as long as the hardware has some basic error-detection mechanisms. (Our own approach to this is described later.) The address size is important because if it is too small for the expected applications, the ensuing problems cannot be completely overcome by software. The PDP-11 I/O architecture is an example of a feature that turned out to be unimportant because it could be completely hidden from users by software.

At a higher level, users of C.mmp seemed satisfied with the overall system performance.

Our ability to support multiprocess algorithms is well established by the performance of the many applications on C.mmp.

We have successfully supported user processes that require real-time response, although this was not one of our major goals.

At the end of the paper we will give some performance figures for an application which runs on several CMU computers, including C.mmp.

Most often cited criticisms of the system were:

Interaction with operating system facilities, in or out of the kernel, is accompanied by a high overhead.

The most serious obstacle to rapid execution of large systems is the limitation imposed on programming by the small PDP-11 address.

Memory contention significantly degrades performance when many processes are accessing the same memory page. This is usually caused by the processes sharing the same code pages.

Memory contention is very serious when using high-performance I/O devices which depend on rapid access to memory during transfers.

The performance bottlenecks are due to a combination of avoidable and unavoidable factors. We were initially distressed at the high operating system overhead (it takes about 500 microseconds to enter and exit the kernel), but we attribute most of it to a lack of experience with the fairly complex features we wished to implement. We are confident that the overhead is not an inevitable result of our protection mechanisms, nor is it due to the hardware design.

Memory contention, caused by several processors trying to access the same memory simultaneously, was a performance concern from the outset of the project. Our simulation studies indicated that its effect would be minimal, but in practice several circumstances conspired to make the problem significant. First, typical large multiprocess applications tend to share the same code among all processes, and this greatly increases the probability of accesses to the same memory. Second, the installation of per-processor caches, which were to handle this code-reference problem, has been delayed due to various resource shortages. Finally, we found that devices such as our disks and drums could not tolerate the long memory access times characteristic of periods of high contention. A software solution to this problem had to be implemented.

The small address problem is serious for large applications which cannot fit within the 64K address space on the PDP-11. Although we could not have avoided this problem, we were guilty of underestimating its significance for the applications which were to run on C.mmp. The problem is considered in more detail later in this paper.

Protected subsystems

In Hydra, the construction of operating system facilities outside the kernel is centered around an abstraction called a *protected subsystem*. A subsystem is, in its basic form, a new object type combined with a set of procedures which operate on objects of that type.

Our experience derives from over twenty working subsystems implementing schedulers (*Policy Modules* in Hydra terminology), files, directories, an I/O device allocator, and a host of other traditional operating system facilities. As

software development continued by diverse users, we were curious to see whether all the required software could be built within the subsystem abstraction, whether such development could be done easily and quickly, and whether the resulting facilities could be easily merged into the user environment.

> The protected subsystems abstraction is very powerful in designing operating system software in a capability environment.
>
> It is easy to design subsystems which are easy to use and which are protected from any interference from software outside the subsystem.
>
> The subsystem structure makes it easy to provide several coexisting and competing facilities.
>
> The subsystem structure is useful for isolating facilities under development or being debugged.
>
> New subsystems are easily incorporated into the standard system.

We think the subsystem concept in Hydra is as useful as the closely-related notion of extended data types has been in the field of programming languages. Part of the original motivation for the subsystem concept was our desire to allow alternate solutions to problems which we could not foresee in a multiprocessor environment. However, we found that subsystems are also very useful in debugging versions of "standard" systems without interfering with users.

Many people at the meeting were critical of the failure to follow up the subsystem design with the software tools which would encourage building subsystems in this new environment.

> Subsystem construction still suffers from being ad hoc, there being inadequate software support for managing the programs, data structures, and documentation which comprise the subsystem.
>
> The development of system software (subsystems) by many different people makes it more difficult to impose any standardization.
>
> Subsystems are less likely to be successful when they attempt to implement traditional (non-capability) systems in traditional ways.

These problems are the result of our not giving the user environment outside the kernel as much attention as we gave the Hydra kernel itself. We consider it one of our worst mistakes and will discuss it more later in the paper.

Scheduling is an example of a traditional operating system function which, in Hydra, is partially implemented outside the kernel by a subsystem called the Policy Module (PM). We thought that providing scheduling policy outside the kernel would allow us to experiment with different specialized strategies for scheduling cooperating processes.

The first Policy Module is a distinguished subsystem for several reasons. First, it was one of the first subsystems built outside the kernel and exhibits many of the mistakes of any first attempt. Second, it is a particularly nice example of our ability to build operating system facilities outside the kernel. Finally, it interacts very closely with the kernel, so the efficiency of the kernel interface is emphasized.

The first Policy Module was operational from 1974 through May, 1977. Our basic evaluation at the meeting was that

> The first Policy Module adequately demonstrated that traditional policy decisions could be made outside the kernel.

In spite of this, many people noted flaws in the implementation which were glossed over in our rush to see if the PM would work.

> Insufficient attention was paid to reliability and throughput in the Policy Module.
>
> The PM-kernel interface turned out to be more complex than we had anticipated.
>
> We included things in the kernel facilities which logically belonged outside; this acted to complicate the kernel interface. [*For efficiency reasons, we implemented in the kernel some facilities which should have been outside according to our philosophy.*

Hence,

> The construction of Policy Modules was not as easy as we had imagined before we actually tried it.

Because we expected a PM to incorporate specific knowledge about the processes it was scheduling, we anticipated having many PM's simultaneously scheduling different sets of processes. Indeed, having several PM's run at the same time was no problem, but again the performance left something to be desired.

> To support multiple Policy Modules, more facilities are needed in the kernel to ensure a fair allocation of processor and memory resources to each Policy Module.

We began to build a second version of the Policy Module almost as soon as the deficiencies in the first were recognized. This design proceeded in parallel with performance improvements to the first PM, and in fact we were running both PM's simultaneously for a short time.

The distributed operating system

Hydra was designed with no master-slave relationship among processors. With the exception of the lowest level of I/O device support, all system tasks may run on any and all processors. An immediate result of this is that we expected

a high degree of parallelism in Hydra and the corresponding need for effective synchronization methods.

There are two notable aspects to our approach to synchronization. First, we decided to synchronize on data rather than code. Every data structure which can be accessed by more than one processor is provided with a lock or semaphore which is used to ensure mutual exclusion.

Second, we provided a range of synchronization primitives, from very fast "locks" to much slower "semaphores." The tradeoff here is the overhead needed to P or V the lock or semaphore against the resources which will be tied up by a process waiting to pass the lock or semaphore. Small data structures which are locked for short periods of time (order 300 microseconds) use locks, which involve a very small overhead (approximately four instructions) when the process does not block. Large data structures, or data structures whose processing may be interrupted for long periods of time (as when waiting for I/O) use semaphores, which tie up fewer resources when blocking is necessary.

The simple, symmetric hardware has permitted a much simpler operating system design.

Hydra hides the processor-device correspondence so well that most of Hydra, and all the software at the user level, is unaware of the actual location of I/O devices.

The symmetric distribution of the operating system has been an unqualified success. We are able to achieve a high degree of parallelism within Hydra, and the system is insensitive about the number of processors available.

The use of asynchronous processes ("demons") to implement system functions resulted in simpler designs and improved performance.

In providing synchronization within the kernel, we believe we profited by locking data structures rather than code.

Our decision to provide several types of synchronization mechanisms gave us much design flexibility.

The natural synchronization primitives and our conscious and constant commitment to a high degree of parallelism has resulted in our encountering few software bugs caused by inadequate synchronization.

We have found that the use of demons to absorb much of the system work load outside the normal computational stream has simplified much of Hydra's design. We might not have used this technique if we did not have so much confidence in our synchronization techniques and our ability to achieve a high degree of parallelism.

Coverage of hardware and software errors

There are times when clouds do have silver linings. From the earliest days of the project we had to contend with unreliable hardware and our own software mistakes; moreover, we could not afford a 24 hour/day operator to reload the system after each crash. Thus we were forced to consider the general problems of software detection and recovery from errors—whether they be hardware or software induced.

When an error is detected by Hydra, we try to answer a number of questions. What was the exact error? Can we tell if it is due to a hardware or software malfunction? If hardware, is the problem repeatable or transient? Have any critical data structures been damaged? If so, can the damage be repaired? Can we eliminate a piece of malfunctioning (or just suspicious) hardware and still run? In all cases, our aim is to keep the system running with as much functionality as possible.

Our probability of detecting an error soon after it has occurred is increased by building error-detection mechanisms into the hardware and software. The CMU-built memory relocation units implement parity checking on every memory byte and on the address bus through the crosspoint switch. Software modules employ redundant representation and other techniques to try to limit the propagation of errors not detected by the hardware.

Recovery mechanisms invoked by the detection of any error employ a "suspect-monitor" paradigm to ensure that a failure in the recovery processor may be detected cleanly. Two processors are always involved; one, the *suspect,* attempts to record the system state at the time of the error; the other, the *monitor,* watches the suspect and assumes control if the suspect is unable to finish. The suspect is always the processor on which the error occurred. The monitor is selected at random from all other processors. There are a number of steps which can be taken during a recovery action depending on the type of error, including removing processors or memories from the system and producing extensive crash dumps for later off-line analysis.

The fault tolerance built into some kernel modules resulted in making them among the most reliable in the system—more reliable than other modules coded by the same programmer without using such techniques.

The software facilities for detecting software and hardware errors and restarting the system automatically have been a big success.

Similar facilities in user software are beginning to be developed and show much promise in improving overall system reliability.

Even though we are proud of our current error-handling mechanisms, we know that system needs more work in this area, particularly in the area of supplying policies to determine which mechanisms should be invoked for different types of errors. While it is true that we can recover from virtually any error by initiating an automatic reloading of the operating system, this is a drastic action we would like to use only in the case of truly catastrophic errors. Unfortunately, the difficulty in pinpointing the exact location of some hardware errors and the difficulty in verifying the consistency of the complex capability data structure has resulted in our classifying almost all errors as "cata-

strophic" in this sense. We are in the midst of redesigning both hardware and software to correct these deficiencies.

Software development methodology

Our initial goals for the Hydra implementation did not explicitly include the notion of exploring a software engineering methodology. Nevertheless, we used a method based on Parnas' "modular decomposition"* and it worked quite well; indeed many of us believe that without it the project would not have succeeded.

The methodology used caused us to divide the units of work (programming tasks) along the lines of the major data structures in the system. A module (and hence a programmer) was responsible for the representation of, and all operations on, a data structure. No one other than the responsible programmer had access to knowledge concerning the implementation details.

Because methodology *per se* was not our major goal we were not fanatical about enforcing the methodology, and were often less precise about the specifications than we might have been. Both the positive and negative aspects of this informal approach are reflected in the following remarks:

> We believe that it is a measure of the success of the modular implementation of the kernel that one full-time programmer can maintain this program which comprises 2000 (listing) pages of source code.
>
> The independent implementation of the modules in Hydra resulted in a lack of any uniform coding style and in some duplicated effort in interfacing to the underlying hardware. The effect was not very serious since all the implementors were highly talented, exhibiting differences in style rather than quality.
>
> Because modules were implemented independently, no one initially had a detailed knowledge of the entire system. This made debugging more difficult and resulted in a difficult transition when Hydra began to be maintained by a single programmer who was not part of the original implementation team.
>
> Coding of the kernel began quickly after the initial design. Some think too quickly.
>
> Loose management coupled with the modularization technique worked well except in promoting a standardization of coding styles.
>
> Information hiding as a modularization technique resulted in coding situations in which information necessary to make a decision was not available.
>
> As Hydra developed and was modified, the original, clean modularization began to break down as new features were added and performance bottlenecks removed.

* Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *CACM*, 15, 12, pp. 1053-1058, 1972.

We still think the modular decomposition methodology is extremely good for structuring large systems. In our experience, breakdown of the modular structure occurs mainly when programmers in the midst of debugging adapt "quick and dirty" solutions which do not preserve modularity.

All but a very small part of Hydra is written in a high-level implementation language, Bliss-11. There seems to be no question that it was possible, indeed advantageous, to write the kernel in Bliss, but there were problems. The Bliss-11 compiler was developed only shortly before the kernel was begun and was an independent research project (investigating compiler optimization techniques). There was some initial friction between the two groups, but both appear to have benefited in the long run.

> The Bliss-11 compiler was designed to compile a slightly modified version of the Bliss-10 language into very compact PDP-11 code. This it does.
>
> The implementors of the Hydra kernel were, and continue to be, a major influence on the addition of new features to Bliss-11.
>
> The facilities of the Bliss-11 language and compiler had a significant influence on the coding of Hydra.
>
> Some of us believe that Hydra could not have been written in this environment without a language of Bliss's caliber.
>
> Bliss-11 preceded Hydra by too short a time. The unreliability of the compiler during its first year of use hindered kernel development.
>
> Compatibility between Bliss-11 and Hydra was a problem. Changes in Bliss-11 sometimes had unfortunate consequences on Hydra code.

We think these comments reflect the close interdependence between a large programming project (Hydra) and the software engineering tools it uses (Bliss-11). Bliss was in a real sense critical to Hydra's development. The need to debug both Bliss and Hydra simultaneously was a necessary burden.

A common measure, albeit a crude one, of a methodology is the productivity of the programmers which used the methodology. By that measure our development strategy worked very well; the average productivity has been about 20 instructions per man-day for kernel code (the typical industrial average for similar code is 5-7 instructions per man-day).

THE FAILURES

Hardware reliability

Hardware (un)reliability was our largest day-to-day disappointment at the time the evaluation meeting took place. The aggregate mean-time-between-failure (MTBF) of C.mmp/Hydra fluctuated between two to six hours, where a failure is defined to be any situation which triggers the recovery actions described earlier. About two-thirds of the failures were directly attributable to hardware problems.

There is insufficient fault detection built into the hardware.

We found the PDP-11 UNIBUS to be especially noisy and error-prone.

Our paging drums were chosen for their predicted performance, but their reliability was so poor that performance was often a moot point.

The crosspoint switch design is too trusting of other components; it can be hung by malfunctioning memories or processors. [*This almost never happens, but when it does automatic recovery is impossible.*]

We made a serious error in not writing good diagnostics for the hardware. The software developers should have written such programs for the hardware.

In our experience, diagnostics written by the hardware group often did not test components under the type of load generated by Hydra, resulting in much finger-pointing between groups. Faulty hardware is often kept in the user system because only Hydra can provoke and pinpoint errors.

Several components of the system have gone through several development cycles, mostly to improve the handling of exceptional conditions, but we are basically limited by the capabilities of the PDP-11 and its UNIBUS. There appear to be two flaws in many of the off-the-shelf components. One of these was mentioned during the meeting: the lack of mutual suspicion. There are a number of ways in which the entire system can be made to fail if one inessential component does not operate according to specifications. The other flaw was not mentioned: the failure to *contain* errors. Once an error has been detected the goal should be to make absolutely sure that the damage won't spread. Many of the standard components, unfortunately, will "complete" an operation even when an error is known to exist; in completing the operation they destroy data, thus making the error unrecoverable.

There is some good news to report, however. Following the meeting, increased emphasis was given to hardware maintenance. As this paper is written (January 1978) our MTBF has increased to about ten hours and many of the hardware problems seem to be settling out.

The small address space problem

The PDP-11 is a 16-bit minicomputer; of particular interest is the fact that this restricts all addresses generated by a user program to be 16 bits long. These 16 bits can be used to address no more than 64K bytes of memory. We refer to this limitation as the "small address problem", or SAP.

Although we were initially aware that the operating system would have to provide some sort of facility for allowing a user to address more than this amount of memory, we did not appreciate how restrictive the 16-bit limitation would be or to what extent circumventing it would affect performance. Our initial impression was that the 16-bit limitation would be offset by the ability to create multiprocess programs—that the typical program organization would be a larger number of processes, each addressing a smaller amount of memory. That impression turned out to be false, as is reflected in some of the comments made at the meeting:

Our initial prediction that programs would be implemented as small subsystems using less than 64K was wrong.

Multiprocess algorithms do not always produce small programs.

Even though programmers are writing programs which execute on PDP-11's, their tasks are CDC 6600-size.

There is nothing good to say about this problem other than that we were pretty much forced into it.

To circumvent this problem, Hydra provides a facility, supported by the hardware, to divide the address space into 8 pieces, each of which is called a "page." The user is permitted to have an indefinitely large number of pages, but to address only 8 of them at any instant. Operating system facilities are provided to allow the user to dynamically designate which of his pages are to be addressable; he does this by associating a page with one of the 8 "relocation registers" maintained by the hardware. Thus, except that the cost of loading is larger, the addressing scheme is very similar to the use of "base registers" on 360-370 style machines. We have found this facility, however, to be less than ideal.

Page boundaries are absolute, and the programmer must always be aware of them.

The problem is in addressing data. There are easy solutions to addressing code segments.

More relocation registers and a smaller page size would reduce but not eliminate the problem.

We believe the problem would exist even if making pages addressable required no overhead.

Because of the performance penalties associated with managing the address space, the inconvenience cannot be hidden from the user through a high-level language:

L*'s ability to allow access to large amounts of memory has been hindered by the short PDP-11 address. [*L* is a list processing language used for the implementation of large systems.*]

It must be emphasized that not *all* programs are affected by the small address space problem:

In practice, most subsystems have no problem fitting into 64K.

Our failure on the small address problem was really one of misappreciating the way in which the machine would

actually be used. The remark above to the effect that many tasks are 6600-size is a telling one. The machine is comparable in size to a 6600 and people want to use it that way. Big problems often imply big data and we failed to appreciate that during the initial design.

The partitionable system

When we first considered the possibility of building a multiprocessor in 1971, the ability to partition it into several disjoint subsystems was on our list of advantages for such architectures. While we are able to partition processors and memory, we are not able to run Hydra in more than one partition.

C.mmp can be partitioned in such a way that some processors and memories can undergo maintenance and run stand-alone diagnostics without interfering with the larger partition running the operating system.

The primary obstacle to running the operating system in two partitions is the money required to provide each partition with an adequate complement of I/O devices and memory.

We do not know how to provide meaningful communication between the capability structures of the two operating systems.

The principal effect of the failure to meet this goal has been that we must allocate disjoint time for users, hardware maintenance, and operating system testing. At present 28 hours each week are reserved for maintenance. This partitioning has been very inconvenient for all concerned, and has certainly impeded progress on several occasions. Yet it seems clear that we have been unwilling to spend the money necessary to solve the problem—thus it seems safe to conclude that the inconvenience has not been debilitating.

(The lack of) human engineering

As we have mentioned in several contexts previously, the human interface to the C.mmp/Hydra system is not well designed. To some extent this resulted from the novelty of the underlying system structure (we couldn't anticipate some of the kinds of facilities that would be needed by users of either a capability-based or a multiprocessor system). To a large extent, however, the failure seems to have been one of having concentrated on the new, innovative aspects of the system and ignoring more mundane issues.

There is a lack of human engineering in the operating system software which interacts directly with a user sitting at a terminal.

It is difficult to pick up the minimal knowledge needed to know how to do useful things at a terminal.

New users tend to have bad first impressions of the system.

We did not realize how much work was required to make a smooth user interface and so did not allocate enough resources for it.

We suspect the user environment would have received more work had the kernel implementors had to use it during their software development. (All kernel development and maintenance has been done on the PDP-10 computer, which has the Bliss-11 compiler and a linker for C.mmp.)

One particular aspect of the human interface is especially interesting—the command language. It seems to be an almost universal phenomenon that people don't like whatever command language they have used in the past. We were no exception. Thus, rather than modeling our command language on any existing one, we chose to strike out in another direction. In particular, we chose to make the command language a (modest) interactive programming language—with declarations of variables, assignments, conditional and looping control constructs, macros, and so on. The power of this approach seems unquestionable, as is reflected by the following remarks. The remarkable thing (to the editors) is the lack of negative remarks during the meeting; the command language usually comes under heavy attack on other occasions.

The Command Language is much more flexible and powerful than the command scanners found on most systems.

The concept of the Command Language as a programming language was good.

The Command Language user on C.mmp is unique in having complete access to the Hydra environment. Subsystems can almost be implemented directly in the Command Language.

Error reporting by the Command Language is poor.

Another aspect of the human interface is the (lack of a) spectrum of programming languages:

C.mmp lacks the wide range of languages available on conventional systems.

The L* system provides its users with a complete environment compatible with that provided on the PDP-10 by its version of L*.

The L* environment does not seem conducive to the construction of subsystems.

The Algol 68 implementation on C.mmp gives users access to the multiprocess capabilities of C.mmp, but does not yet provide access to capabilities or the Hydra protection environment.

The fact that most subsystem development takes place partially on C.mmp and partially on the PDP-10's (which

have Bliss-11 compilers) is not a severe hindrance now that smooth communication facilities exist between the machines.

It is interesting (to the editors) that the word "baroque" was not used during the meeting; in other contexts it often is. Several features of Hydra and its subsystems do exhibit "second-system-itis". There are things which are more general, and more complicated, than necessary.

Project management

The C.mmp/Hydra project was not a large project by most standards; there were never more than about 15 people, mostly students, working on the project at any one time. Nevertheless we made a number of errors which can only be classified as failures in the management of the project; taken together, these errors constitute one of our largest failures.

Among our errors is a classic! Because the hardware and Hydra structures were new and exciting, we tended to focus on them to the exclusion of the more mundane things which also determine the ultimate utility of any system. This point recurred in many of the points raised at the meeting:

The manpower allocated to the Policy Module was inadequate. In fact this was true of all software outside the kernel.

The failure to stress reliability and performance in the first PM was a mistake.

The user environment was ignored at first because of our natural preoccupation with the Hydra kernel and the research problems it embodied.

We underestimated how much work would be involved in constructing the user environment.

We have a much better idea now about the proper structure (or at least an adequate one) of the user environment than we did when we began building the first subsystems. Implementing basic concepts such as "jobs" and "terminals" in nonpriviledged software has subtle design and reliability implications which we are just now appreciating.

The management style used throughout the project was informal. There were very few memos, formal design reviews, or the other mechanisms of tight management control. In most ways this felt appropriate to the academic environment and the high caliber of the individuals involved. It led to a number of problems, however, and the consensus of the meeting was that the management had been too loose. This is especially evident in the comments relating to a lack of formal specifications and the lack of uniform documentation and coding standards.

The fact that the Hydra implementors did not have to use C.mmp for software development contributed to the neglect of the user environment.

The lack of detailed hardware specifications hindered the parallel development of hardware and software but not the end result.

Software was occasionally developed which took advantage of unspecified "features" of the hardware, making them difficult to change.

Loose management coupled with the modularization technique worked well except in forcing standardization of coding styles.

We should not have depended on graduate students for complete software development for so long. Graduate students cannot keep deadlines reliably and are not tied to the project. [*Furthermore, we feel that Ph.D. students should not spend an inordinate amount of time doing the standard programming chores which characterize any attempt to bring up a complete operating system.*]

Another class of management errors relates to what might be termed "public relations." Being academics, we instinctively react somewhat negatively to the "attention-getting" aspect of PR, forgetting that its "information-providing" function is absolutely necessary. In a number of ways we failed to make information available publicly.

Our problem is basically public relations—performance measurements indicate we have a winner on our hands.

The lack of a smooth user environment was a deterrent to new users which could form the foundation of a happy and vocal user community.

Since Hydra was not easily accessible to people outside the department, we could not adopt a "try it and see" attitude.

Documentation is needed to encourage use internally and generate credibility externally.

A DATA SAMPLER

The previous section concludes our report of the meeting. Since the body of the report contains many subjective and unsubstantiated comments, we decided to include a few examples of the kinds of data on which these comments are based. We have chosen two examples: (1) a study of the effect of the small address problem on a specific user program, and (2) a study of the contention for locks in the Hydra kernel.

A study of the small address problem

The program used in this study of the SAP is HARPY. HARPY is a speech-understanding system which has been implemented on all of the departments major computers: C.mmp, a stand-alone PDP-11 running under UNIX, and the PDP-10 (both KA10, circa 1967, and KL10, circa 1976, processors are available in the department). Since HARPY ex-

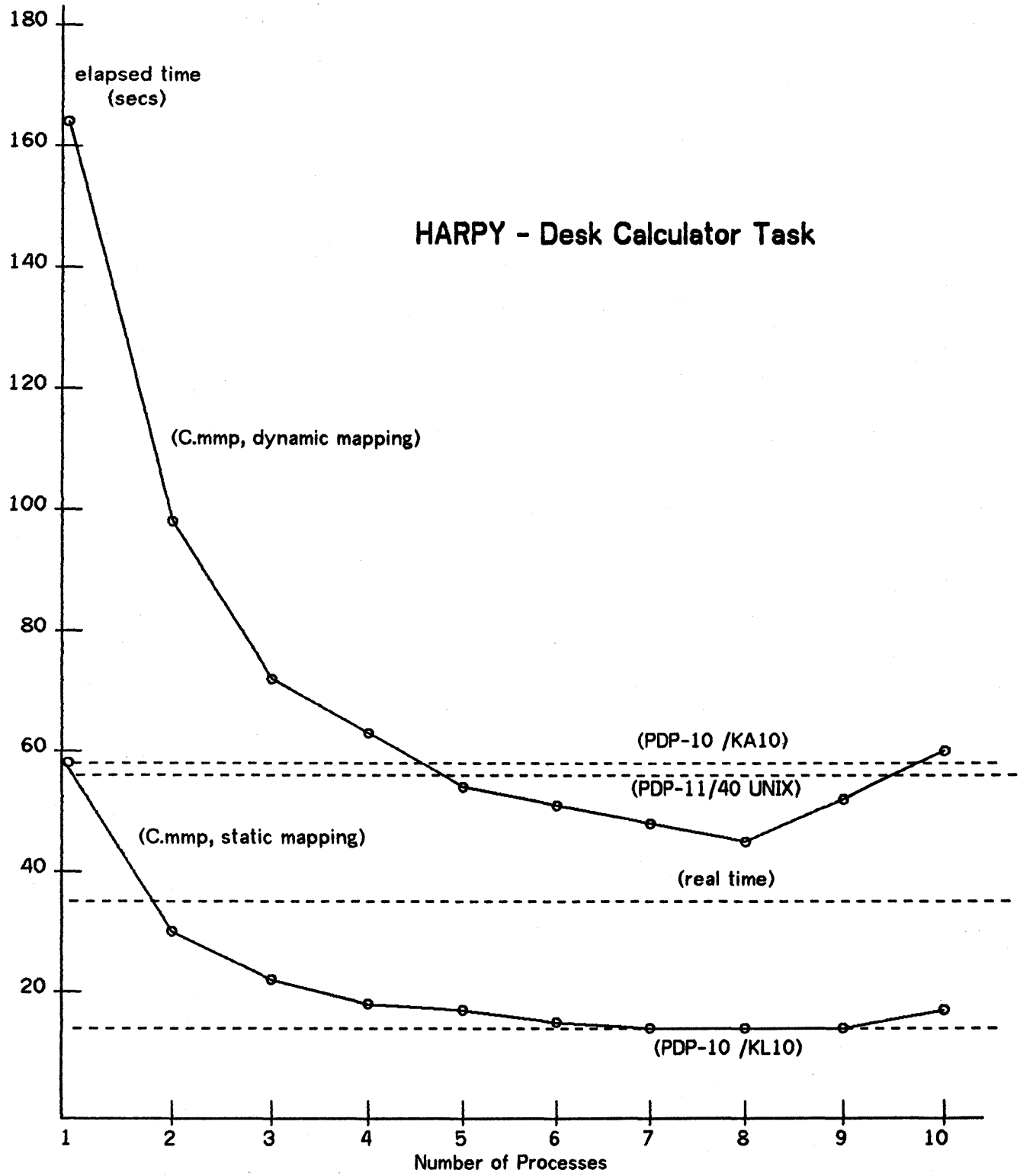


Figure 1—A look at the small address problem

ists on all these machines, it makes a convenient benchmark. (We should point out that HARPY is not necessarily the best application for C.mmp, nor are the HARPY implementations on C.mmp known to be optimal.)

Figure 1 summarizes the data obtained from a series of experiments with HARPY working on a rather small task, namely a voice-input desk calculator that has a 37 word vocabulary.

The horizontal dashed lines represent the performance of single-process implementations of HARPY on the departments uniprocessors. The solid curves represent the performance of two implementations on C.mmp, both of which can utilize any number of processes.

The two HARPY versions on C.mmp differ in their assumptions about the addressability of data. The "static mapping" version knows that all of its data is always addressable, while the "dynamic mapping" version expects to have to do some mapping of relocation registers in order to address the data. In this second version, it must be realized that, in fact, all the data is addressable, and thus no operating system overhead is involved. (The overhead is HARPY checking to see if relocation is necessary—it never is.)

This type of data dramatically illustrates the effect of the SAP on performance—it costs nearly a factor of three in this example. The effect on programming difficulty is at least as great, but is not so easy to illustrate.

Note that the one-process, static mapping version of HARPY runs very nearly as fast as the version running under UNIX, even though the C.mmp version has all the necessary mechanisms for multiprocessing. We think this indicates that the synchronization primitives (spinlocks in shared memory) do not contribute much overhead in this application.

Also note that little improvement in performance is seen beyond three or four processes. This is simply due to a lack of work to do—the small vocabulary simply isn't complicated enough to keep the processors busy. On larger vocabularies we typically see noticeable improvement out to eight processes. The upturn in the curves towards the end is due to the fact that all the faster PDP-11/40 processors are in use. As soon as one PDP-11/20 is used, the whole assemblage of processes slows down. This is because the particular decomposition of the algorithm limits the speed to that of the slowest process.

A study of kernel lock contention

One of the largest potential bottlenecks in a distributed operating system is contention for locks on shared data structures. The hardware monitor has been used to study this; the types of results obtained are shown in Figure 2.

In this study, three programs with seemingly different demands on the system were run while the hardware monitor measured the activity on one processor. The data is illustrative only, since no claim is made that the programs in any way represented a "typical" system load.

The principle result is that it seems we spend consistently less than 1 percent of the time blocked on locks. We do not yet have any measurement of the time lost due to blocking on semaphores.

Static	Program 1	2	3
Total time of measurement (millis)	17393	32924	20255
Number of different locks detected	53	79	181
Average time inside a critical section (micros)	279	378	279
Total number of lock operations	2955	504	4360
Percent of locks which blocked	5.5	11.7	6.1
Percent of time spent in kernel code	61.8	16.9	37.7
Percent of time spent in blocked state	.29	.83	.74

Figure 2—A study of kernel lock contention

CONCLUSIONS

The C.mmp/Hydra project has reached the point at which many of its most interesting and important results will emerge. With a growing user community, increasing reliability and a smoother user interface, we are in a position to gether data on various aspects of system performance under real loads. This data will augment that already collected on isolated algorithms to provide a comprehensive picture of C.mmp/Hydra performance. Along the way to constructing the current system we managed, in our opinion, to do some things well and some things not so well. This paper has been our attempt to report those opinions in the hope that others may benefit from our experiences.

ACKNOWLEDGMENTS

A large fraction of the faculty and staff of the Computer Science Department at CMU have been involved with C.mmp and Hydra over the past five years—as designer/implementors, as users, or as constructive critics. We are deeply indebted to all of them. We are especially indebted, however, to those who participated in the meeting that is reported here:

Hardware:	Bill Broadley, Jim Teter
Hydra:	Sam Harbison, Dave Jefferson, Roy Levin, Hank Mashburn, Fred Pollack
Non-kernel OS:	Bill Corwin, Rick Gumpertz
Performance Evaluation:	Sam Fuller
Major Users:	Anita Jones, Bruce Leverett, Pete Oleinick, George Robertson
Others:	Joe Newcomer, Bill Wulf

We would also like to thank Guy Almes, Peter Schwarz, and the NCC '78 referees for their helpful suggestions for this paper.

C.mmp/Hydra BIBLIOGRAPHY

This bibliography includes references to papers, articles, and theses related to the design, development, and measurement of C.mmp and Hydra. There are numerous internal documents and memos which are not included.

1. Almes, G. and G. Robertson, "An Extensible File System for Hydra," Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, Pa. 1978. (This paper will appear in the *Proceedings of the Third International Conference on Software Engineering,* 1978.)
2. Bell, C. G., W. Broadley, W. A. Wulf, and A. Newell, "C.mmp: The CMU Multiminiprocessor Computer: Requirements and Overview of the Initial Design," Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., August 1971.
3. Bhandarkar, D. P., "Analytic Models for Memory Interference in Multiprocessor Computer Systems," Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., September 1973.
4. Bhandarkar, D. P. and S. Fuller, "Markov Chain Models for Analyzing Memory Interference in Multiprocessors," *ACM/IEEE First Annual Symposium on Computer Architecture,* Dec. 1973, pp. 231-239.
5. Cohen, E., "Problems, Mechanisms and Solutions," Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., August 1976.
6. Cohen, E. and D. Jefferson, "Protection in the Hydra Operating System," *Proceedings of the 5th Symposium on Operating System Principles,* Austin, Texas, Nov. 1975, pp. 141-160.
7. Fuller, S. and P. Oleinick, "Initial Measurements of Parallel Programs on a Multi-miniprocessor," *IEEE CompCon'76,* September 1976, pp. 358-363.
8. Fuller, S., "A Cost/Performance Comparison of C.mmp and the PDP-10," ACM/IEEE Symposium on Computer Architecture, Jan. 1976.
9. Fuller, S., Swan, R. and W. A. Wulf, "The Instrumentation of C.mmp: A multi-(mini)-processor," *IEEE CompCon'73,* 1973, pp. 177-180.
10. Fuller, S. H., and D. K. Stevenson, "The Performance Monitor for C.mmp," *11th Annual Allerton Conference,* Urbana, Illinois, October 1973.
11. Fuller, S. H., "Recent Developments in Multiprocessor Computer Systems," *CALCOLO,* Vol. XII, No. 1, June 1975, pp. 35-58.
12. Jones, A. K. and W. A. Wulf, "Toward the Design of Secure Systems," *Software—Practice and Experience,* Vol. 5, 1975, pp. 321-333.
13. Levin, R., E. Cohen, W. Corwin, F. Pollack, and W. A. Wulf, "Policy/Mechanism Separation in Hydra," *Proceedings of the 5th Symposium on Operating System Principles,* Austin, Texas, Nov. 1975, pp. 132-140.
14. Marathe, M., and S. H. Fuller, "A Study of Multiprocessor Contention for Shared Data in C.mmp," *ACM SIGMETRICS Conference,* Washington, D.C., December 1977.
15. Newcomer, J., E. Cohen, W. Corwin, D. Jefferson, T. Lane, R. Levin, F. Pollack, and W. Wulf, "Hydra: Basic Kernel Reference Manual," Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., 1976.
16. Newell, A., P. Freeman, D. McCracken, and G. Robertson, "The Kernel Approach to Building Software Systems," Computer Science Research Review 1970-71, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., 1971.
17. Newell, A., D. McCracken, and G. Robertson, "L*: An Interactive, Symbolic Implementation System," Department of Computer Science Technical Report, Carnegie-Mellon University, October 1977.
18. Newell, A. and G. Robertson, "Some Issues in Programming Multi-Mini-Processors," in *Behavior Research Methods and Instrumentation,* Vol. 7, No. 2, March 1975, pp. 75-86.
19. Oleinick, P. H., and S. H. Fuller, "The Implementation and Evaluation of a Parallel Algorithm on C.mmp," Department of Computer Science Technical Report, Carnegie-Mellon University, December 1978.
20. Reid, B. K. and J. Newcomer, ed., "The Hydra Songbook—A Vigilante User's Manual," Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., October 1975.
21. Reiner, A., and J. Newcomer, ed., "Hydra User's Manual," Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., August 1977.
22. Strecker, W. D., "An Analysis of the Instruction Execution Rate in Certain Computing Structures," Ph.D. Dissertation, Carnegie-Mellon University, 1971.
23. Wulf, W. A. and C. G. Bell, "C.mmp—A Multi-mini-processor," *Proceedings of the Fall Joint Computer Conference,* 1972, pp. 765-777.
24. Wulf, W. A. and R. Levin, "A Local Network," *Datamation,* February 1975.
25. Wulf, W. A., "Reliable Hardware-Software Architecture," *Proceedings of the International Conference on Reliable Software,* Los Angeles, 1975.
26. Wulf, W. A., E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "Hydra: The Kernel of a Multiprocessor Operating System," *CACM* 17, 6, June 1974, pp. 337-345.
27. Wulf, W. A., R. Levin, and C. Pierson, "Overview of the Hydra Operating System," *Proceedings of the 5th Symposium on Operating System Principles,* Austin, Texas, Nov. 1975, pp. 122-131.

A design methodology for user oriented computer systems

by C. V. RAMAMOORTHY and G. S. HO
University of California
Berkeley, California

INTRODUCTION

A user oriented computer system is a computer system specifically designed to meet the users' requirements. With the advances in solid state technology and the emergence of microprocessors, system designers are able to design specialized systems to satisfy the users' requirements. This trend has led to an era of user oriented design. However, current approaches to the design of computer systems and their evaluation, unfortunately, are based primarily on experience and intuition. The specification, design, implementation, and evaluation of large embedded computer systems, such as the air traffic control systems, the patient monitoring systems and the ballistic missile defense systems (BMD), are very expensive, difficult to test adequately, slow to deploy, and difficult to adapt to changing requirements.[1] The major cause of these problems is the largeness of the systems. The activities of the systems are so varied and so complex that they are beyond the grasp of a single individual. For example, the BMD systems include, besides the data processing subsystem, the radar and missile subsystems. Each of these subsystems requires special expertise to design, implement, and enhance its operations. As a result, each subsystem is usually developed and maintained by a group of experts who have little knowledge of the other subsystems. This produces great difficulties in synchronizing and optimizing the development process. One common problem has been that some final decisions on primitives (essential system characteristics) are made in one subsystem, generally without considering the overall system requirements. These early commitments bias the development process and force and restrict the choices of the other primitives to accommodate them, which, in turn, impose undue constraints on design freedom and reduce the flexibility during integrating and interfacing the system. As a result, the development process is more expensive and time consuming than it should be, and the design that is obtained is usually far from optimum.

Another problem faced in the development of large computer systems is the ever changing system environment. When the system application changes or the technology changes, the system has to be modified to adapt to the changes. However, more too often, systems are designed without taking into account the provision for future changes. When the system evolves, the changes are incorporated into the system in a very disorganized manner. As a result, the unstructureness (or the entropy) of the system increases enormously[2] and leads to a regenerative, highly non-linear, increase in the effort and cost of system maintenance.[3] In addition to this, the reliability and the integrity of the system are also jeopardized greatly.

In large scale critical real time systems, such as the nuclear reactor control systems, the development process is further complicated by the real time constraints and the criticalities of the systems. These systems must perform all the required functions correctly within the given time limits, otherwise a large penalty has to be paid (for example, large loss of life due to a nuclear explosion). However, these systems cannot be tested in its real operational environments. As a result, system validation has to rely heavily on analysis and simulation. Due to the high complexity of the systems, exhaustive testing will be impossible. The only solution is to design the system in an orderly way so that both validation and testing can be done efficiently.

In this paper, a systematic design and development methodology for user-oriented data processing systems (DPS's) is presented. The methodology will provide guidelines for the systematic design and construction of DPS's so that the users' requirements are satisfied if there exists a feasible design under the given constraints. However, it must be emphasized that it does not mean the whole design process can be automated. The engineering decisions will often be very complex and dependent on the experience of the designers. The design methodology will provide design laws and analysis tools to help the designers in making design decisions, trade-offs and predicting the consequences. By following the guidelines given by the design methodology, the very complex design process will be simplified and we will be able to develop reliable, effective, modifiable systems with low costs and lead times.

Characteristics of the methodology

The philosophy behind the methodology is based on hierarchical modelling of a DPS. The objective of establishing

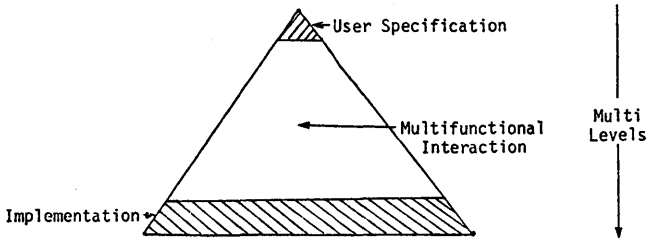


Figure 1—Hierarchical design

the hierarchy is to map the overall system requirements successively into lower levels of finer detail. At the top level in the hierarchy, the requirements described are abstract and the coupling between various attributes and associated functions may be loose. As we proceed down the levels, the characteristics of various functions and the attributes are elaborated and become more specific, Figure 1. The use of abstraction at the top level allows a designer to initially express the system requirements in a very general manner and with little regard to the details of the design and implementation. These initial system requirements are then refined in a step by step manner by gradually introducing more and more details (e.g., constraints and attributes) of the system. This combination of abstraction and stepwise refinement enables the designer to overcome the problem of complexity inherent in the construction of a complex DPS by allowing him to concentrate on the relevant aspects of his design incrementally at any given time, without worrying about other details. By this hierarchical approach, the assumptions and decisions made throughout the design process can be traced systematically and any revisions or modifications of the design as a result of the design development can easily be incorporated. In summary, the design methodology will:

(1) guarantee that the architecture (statement of need, system objectives, and constraints) of the problem will be preserved.
(2) support orderly evolution of the system satisfying the constraints such as performance, reliability, etc. without major revisions.
(3) provide formal (mathematically rigorous) basis for the approach allowing precise evaluation of completeness, consistency and correctness of requirements at any level of definition.
(4) represent effectively and efficiently the decision making constraints of a DPS by a specification language.
(5) provide design attributes and documentation for evolution (growth and modification) so that changes can be made without reconsidering the whole design process.

Overview of the methodology

The design methodology proposed here can be broken down into four successive phases, Figure 2:

(1) requirement and specification phase
(2) design phase
(3) implementation phase
(4) evaluation and validation phase

The requirement and specification phase starts with some (possibly incomplete, vague, and informal) users' require-

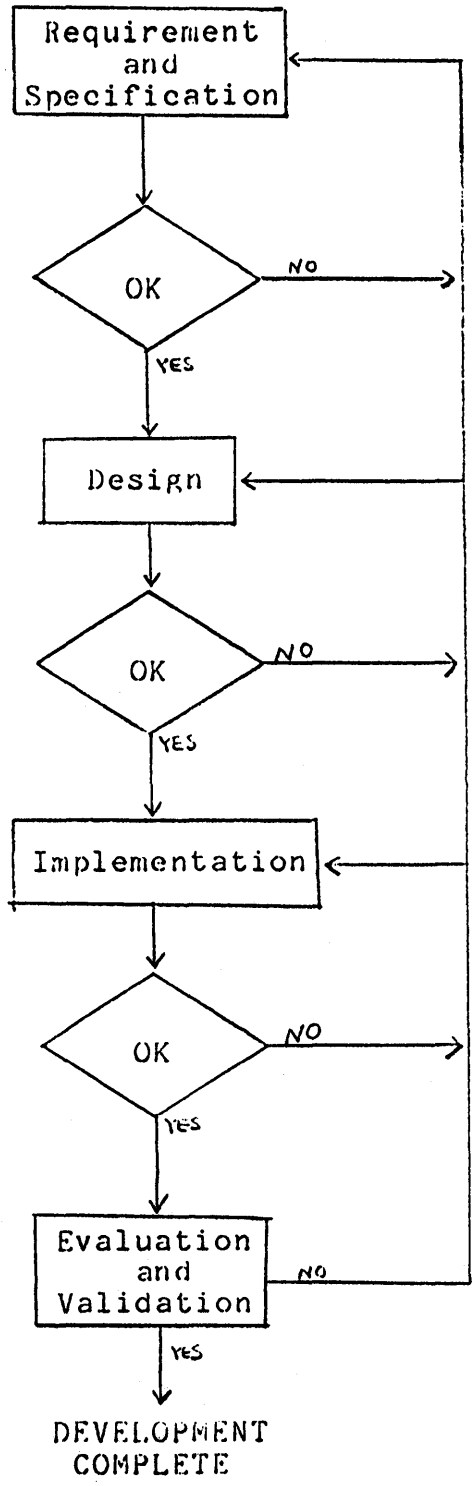


Figure 2—Proposed design methodology

ments that approximate the desired system, and finishes when the modified and elaborated requirements have been formally encoded and tested to the satisfaction of the system engineers and the "customers". This is the most difficult but also the most important step in the development process. Experience has shown that many design failures are due to either ill-defined (inconsistent and unclear) requirements or misinterpretation of the original problem statement. These account for up to 85 percent of requirement errors.[4,5] In order to avoid the above mistakes, the concept of a closed system is used. The users and their environment are considered together during the requirement and specification process. Requirement elaboration is done jointly in the environment and system models. By doing this, more complete and consistent requirements can be produced.

The design phase starts with the requirement specifications and finishes when the system specifications are produced. The objective is to organize and optimize the system in a well formed structure. It involves a hierarchy of decompositions and partitioning of the system into subsystems. Decomposition is the process of dividing the system into several levels of components and subcomponents, and partitioning is the process of grouping these components and subcomponents into subsystems so to minimize the amount of interactions and to satisfy the performance and reliability constraints of the system. After each decomposition and partition step, the subsystems are verified to be consistent to the original system. Any discrepencies and mismatches are corrected before they can propagate into the next level. After the design process, the system functions will be well specified and will be ready for implementation.

The implementation phase takes the system specification and develops the system architecture. It then maps the system functions into either hardware or software functions. It is only at this step that physical constraints and technology come into consideration.

The final step is the evaluation and validation of the system. This phase uses the bottom up validation approach. It takes the final design and ensures that the system meets the original requirements. This step uses both analytical modelling and simulation. Mistakes or unfulfilled requirements are traced back to the source of the error. The system is then redesigned from that point. Since the system is broken down hierarchically, only the subsystems affected by the error and therefore only those that are stemming from the error point have to be redesigned.

It should be mentioned that the development process is not a straight top down process. Tests and checks are conducted throughout the development process. Whenever errors are found, the design is backed up to the previous level. Therefore there is a feed-back path from each development process back to the previous one as indicated in Figure 2.

DESIGN METHODOLOGY

The design methodology is aimed at satisfying a broad spectrum of data processing applications including real time applications. Its primary objective is to develop reliable, effective, modifiable systems with low cost and lead time. In the following sections, the requirement and specification phase and the design phase of the methodology will be discussed in detail. The implementation and validation phases are too technology and architecture dependent and are beyond the scope of this paper.

REQUIREMENT AND SPECIFICATION PHASE

The requirement and specification phase starts with informally specified users' needs and elaborates on them to generate the formal system requirement specifications. These specifications are used for two purposes: (1) as a problem definition for the design process, (2) as a means against which an implementation can be validated. The success in the development of the data processing system greatly depends on the correct interpretation of the requirements in the specification phase. However, these requirement specifications usually suffer from many ills: they are often designs rather than a statement of need; they are usually incomplete and are expressed in an ambiguous language (English); they are difficult to verify and hence are often incorrect, conflictive within themselves (inconsistencies); they are difficult to test and if one wants to modify them, it is difficult to locate and accurately modify all affected areas. In order to rectify the above problems, a systematic procedure is used to generate correct, consistent, complete, traceable, design free and feasible requirements. Formal definitions of the above terms can be found in Reference 6.

In this methodology, the requirement and specification phase consists of four major steps, Figure 3: (i) requirement elaboration, (ii) requirement specification and attribute formulation, (iii) process definition, and (iv) verification of requirements.

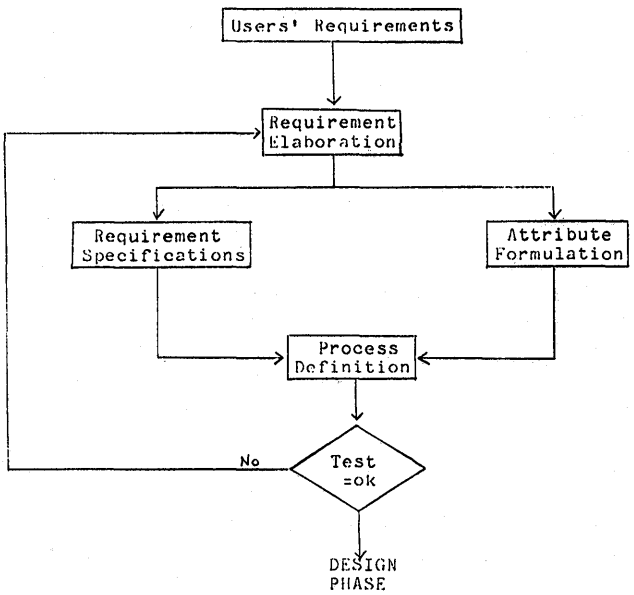


Figure 3—Requirement and specification phase

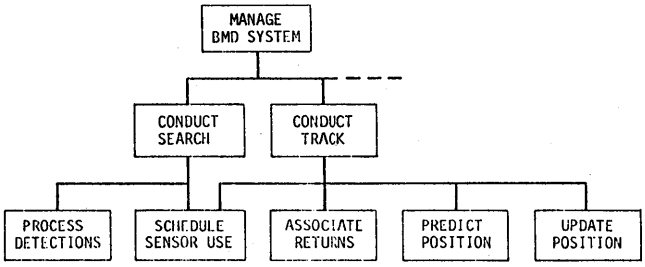


Figure 4—Functional decomposition of BMD management system

Requirement elaboration

The requirement elaboration step can be considered as a problem understanding stage. The requirement engineers investigate in great detail the users' needs and develop clear and precise requirements of the system. In this step, the closed system approach is chosen. In a closed system, the designed system and its external environment are considered as a single entity. All activities in the closed system are investigated in a homogeneous way. After the requirement engineers have fully understood the operations of the closed system, the data processing requirements and the attributes of the computer system can be derived from the behavior of the computer system. These overall system requirements can then be decomposed into finer detail according to their functional, logical or precedence relationships. For example, the BMD management system can be decomposed functionally as shown in Figure 4. Test experiments can also be generated from the behavior of the environment to verify the proper operation of the designed system. In this way, more complete and consistent requirements can be generated.

Requirement specification and attribute formulation

From the required behavior of the system, the users' original objectives can be formally expressed in the system requirements and system attributes.[1] The system requirements are the objectives and constraints which the system must satisfy. Any system which meets the requirements is a candidate solution to the users' problem. Attributes, on the other hand, specify either options or evaluation criteria for qualitative comparisons of competing systems that meet the system requirements. They are used to specify the preferences of the users. The generation of these system requirements and attributes is the requirement specification and attribute formulation step. As pointed out previously, one of the greatest problems in requirement specifications is the misinterpretation of the original system requirements. A plausible solution is to use dual specification teams to develop the system specifications from the requirements independently as in the development of critical real-time software for nuclear power plants.[7] The two specifications are then compared and discrepancies are resolved to the satisfaction of both teams. By this dual specification approach, most of the errors due to ambiguities and misinterpretations can be corrected before they can propagate into the next phase.

Requirement specification

The system requirements can be broken down into two categories: (i) functional requirements, and (ii) performance requirements.

Functional requirements specify the input (stimulus) and output (response) relations of the system. These input to output mappings can be expressed vigorously in mathematical formulas[8] or less formally in a specification language.[9,10] Mathematical formulation defines explicitly the ranges of input and output domains and exhibits the required system functions by tabulation or a set of mathematical expressions. This allows formal consistency and correctness proofs of the system. However, this method is greatly limited by the ability of the requirement engineers in formulating the mathematical functions. In real-world situations, the problems are usually so complex that pure mathematical formulation is impossible. Therefore in this methodology, the approach of using a specification language is chosen.

A specification language is a syntactically and semantically well defined language possibly intermixed with mathematical equations. Its whole purpose is to provide an efficient and effective medium for defining the functional requirements. Several specification languages have been developed previously.[5,9–12] This paper does not intend to develop a new specification language. We will choose a specification language and express the functional requirements in it. In choosing the specification language, the constructability and comprehensibility of the language must be evaluated carefully. The specification language must be able to express the functional requirements efficiently and be easily understood by the customers and the requirement engineers. It must be able to specify the system requirements unambiguously and provide capabilities of performance specifications in the case of real time system. The language should be amenable to both static (hierarchical relationship, data definition, etc.) and dynamic (control flow and data flow) analyses. Finally, it should be backed up by a specification data base management system and powerful graphical supports to provide easy and efficient accesses to the designed system.

Performance requirements specify the functional effectiveness of the system. They include the input and output rates, response time, accuracy etc. There is essentially no notion of completeness as far as performance requirements are concerned. The requirement engineers have to work closely with the users to ensure that all the important performance aspects of the system are captured in the specifications. At this point, it can be noted that there are still some uncertainties and vagueness in the methodology. However, these are unavoidable as design in general is a wicked problem.[13] There is no stopping rule and definite test to the solution of a wicked problem. One stops only because one has run out of resources, patience, etc. Therefore, the best

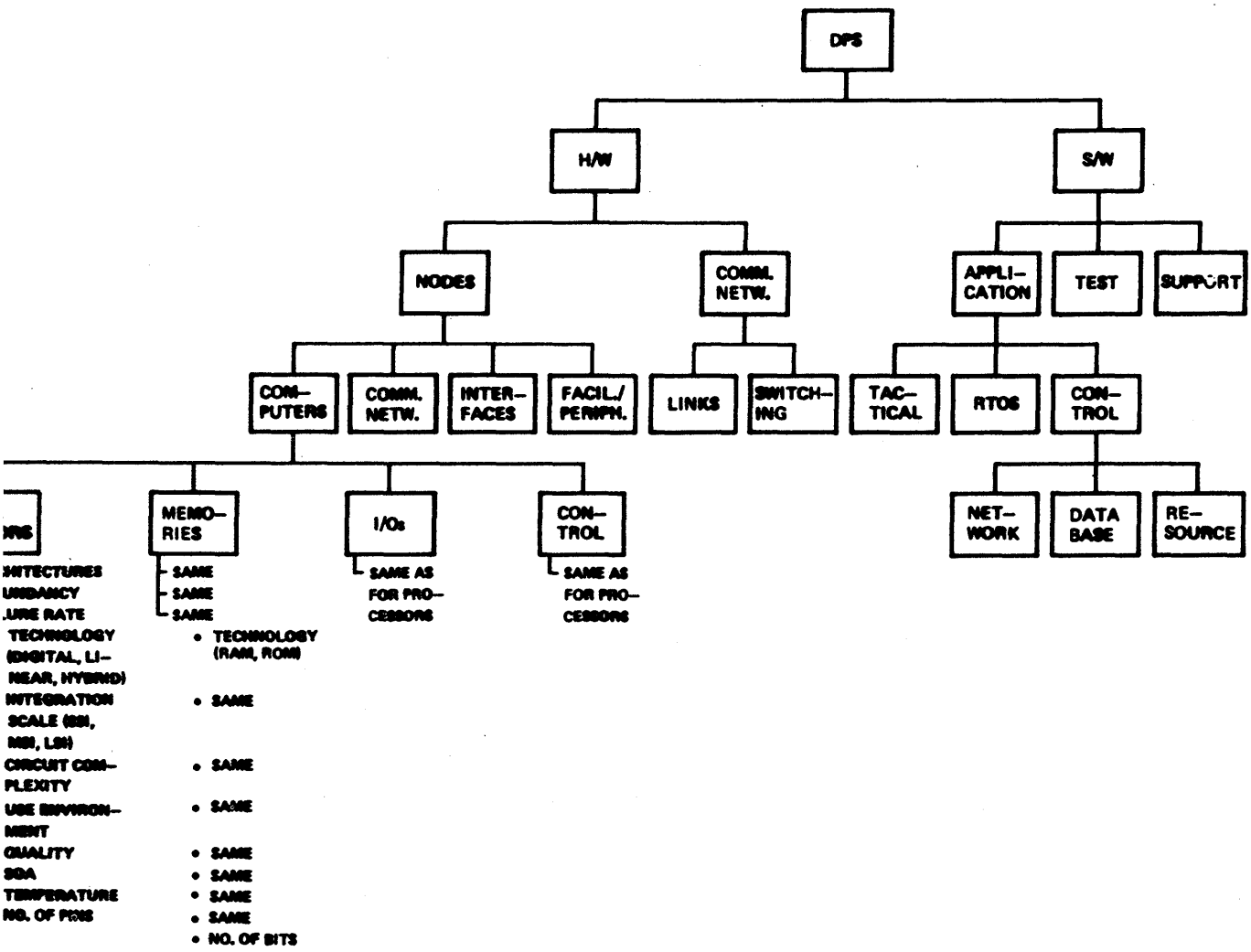


Figure 5—Reliability payoff tree

we can do is try to provide guidelines in the development process to improve the quality of the product.

Attribute formulation

Attributes represent the customers' preference on the designed system, e.g., cost, reliability, availability, flexibility, expandability, reconfigurability etc. They are the evaluation criteria used to determine which characteristics make one system desirable over another system even though they may both meet the requirements. Usually, the system attributes are interdependent on each other and they may compete and interact with each other. In order to clearly specify the preferences of the customers, the system attributes are weighted or ranked according to the customers' preferences. They are then expressed in a utility function[14,15] or in a payoff tree.[16] Utility function has been used extensively in economics and in business management. It measures formally the degree of satisfaction of the customers in terms of the utility of the system. Once the utility function of a system has been formulated, the optimal system configuration can be determined by optimizing its utility. However, due to the inherent complexity of a large computer system and the high interdependency of its attributes, the formulation of its utility function can be very difficult. In such cases, the less formal approach of using payoff tree is preferred. It involves first generating the payoff trees of the system attributes. For example, the reliability payoff tree is shown in Figure 5. The payoff measure interrelationship can then be generated, Figure 6. These payoff trees show clearly to the designers the tradeoffs involved in the attributes. Based on these payoff trees, trade-off decisions in later stages of the development process can be facilitated. This approach is being used in the development of the BMD system.

Process definition

The process definition step accepts inputs from the requirement specification and attribute formulation step and

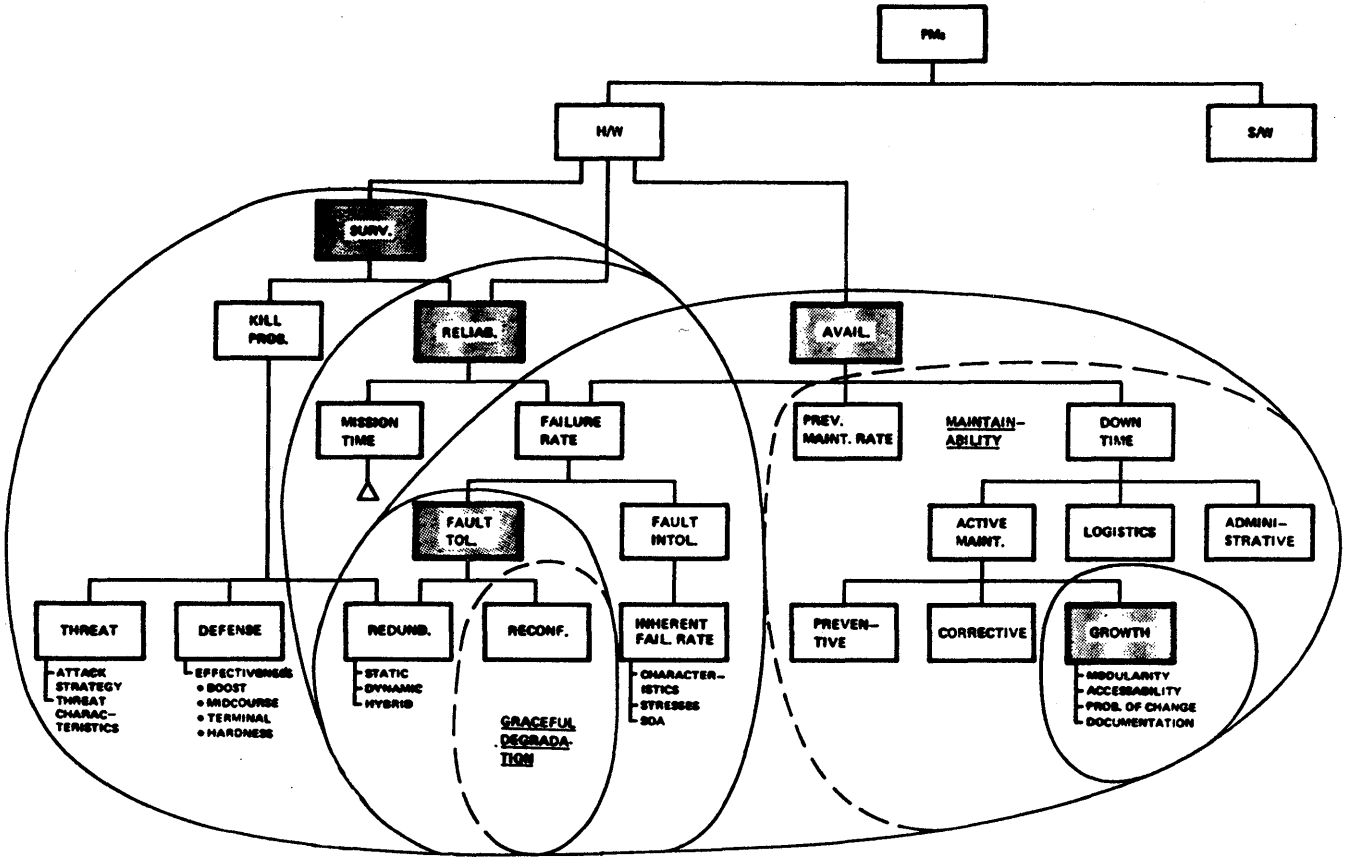


Figure 6—Payoff measure relationships (Note: Taken from Reference 16.)

identifies major functions to be performed. First the input stimulus and the required responses are characterized. This involves stating the form of the input and output signals. They may be mechanical, electrical, optical, etc. From this, the required I/O processes can be defined. For example, in an air traffic control system, it contains the radar control process, the graphic display process and the interactive I/O process.

In parallel to the formulation of the I/O processes, the functional requirements can be decomposed into data processing requirements, communication requirements, precedence constraints, etc. Similarly, the performance requirements can be decomposed into resource requirements, scheduling requirements, etc. Based on these requirements and the attributes defined previously, the information flow and control flow of the system can be modelled and analyzed to identify the major operations to be performed and their locations of occurrence. From these analyses, the system processes required to perform the above functions can be defined. These process definitions state precisely the function of the processes, the resources required by the processes, and the interaction between the processes. At this stage, the virtual system is formed and is ready for verification. For example, in the formulation of a tangent function, it can be decomposed functionally into a sine function, a cosine function and a division operation, Figure 7a. The attributes of the function (like accuracy, delay, output range, cost, etc.) can then be extended to become the attributes of the component functions, Figure 7b. In this manner, the virtual system is formed.

Verification of requirements

In this step, the processes of the virtual system is verified to meet the original users' requirements. As the system is developed hierarchically, the specifications of one level are the requirements of the next level. To verify the correctness of the virtual system, we only have to verify the consistency between the specifications and requirements between consecutive levels. This simplifies the verification process a lot and if an analysable specification language is used, the consistency can be verified automatically. In addition to this, test experiments generated automatically[17] or formulated from the environment in the requirement elaboration step can be used to check the quality of the virtual system. If the tests are not acceptable, necessary changes are incorporated into the requirement specifications. The affected processes will be updated and the corrected system will be tested again.

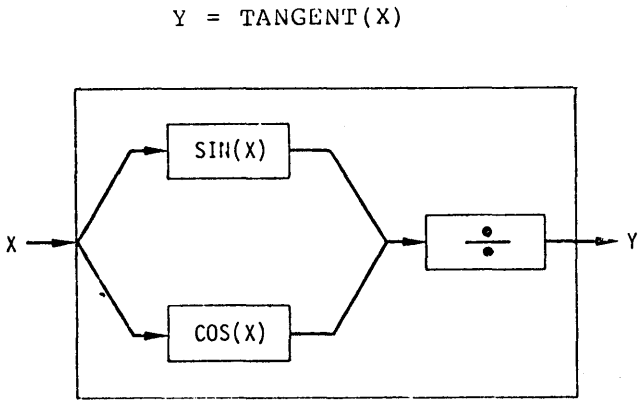


Figure 7a—Functional Decomposition

DESIGN PHASE

The design phase starts with the defined processes which are the output of the requirement and specification phase. The major steps involved in the design phase are process decomposition and partitioning, functional specification and finally verification, Figure 8. Basically, the design phase requires a provision to trace the system requirements through all levels of design, and a means of assessing trade-offs at the functional level and comparing design alternatives.

In the process partitioning phase the designer will decompose the system into a network of interacting tasks. The motivation for such partitioning is to increase modularity and testability and to decrease the interferences between processes.[18] This partitioning is based on the philosophy of hierarchical decomposition of the system into progressively more detailed networks of processes at different levels of abstraction. The hierarchical approach permits partial ordering and allows the designer to isolate interacting and non-interacting parts. The partitioning criteria and approaches depend on the level at which the partitioning is carried out and upon the nature of the logical aspects involved, for example functional decomposition, algorithmic decomposition and attributes decomposition. Several techniques can be adopted to perform partitioning at various levels. Our approach to this problem of partitioning is discussed in the following subsections.

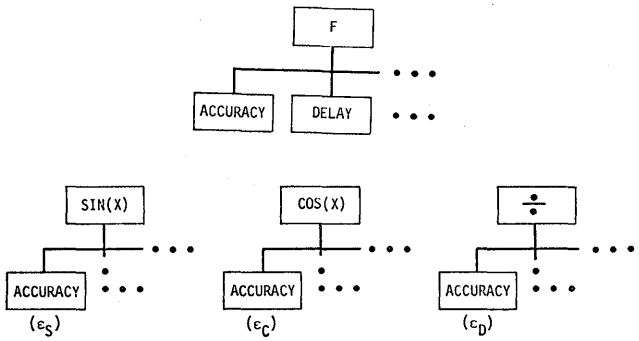


Figure 7b—Attribute decomposition

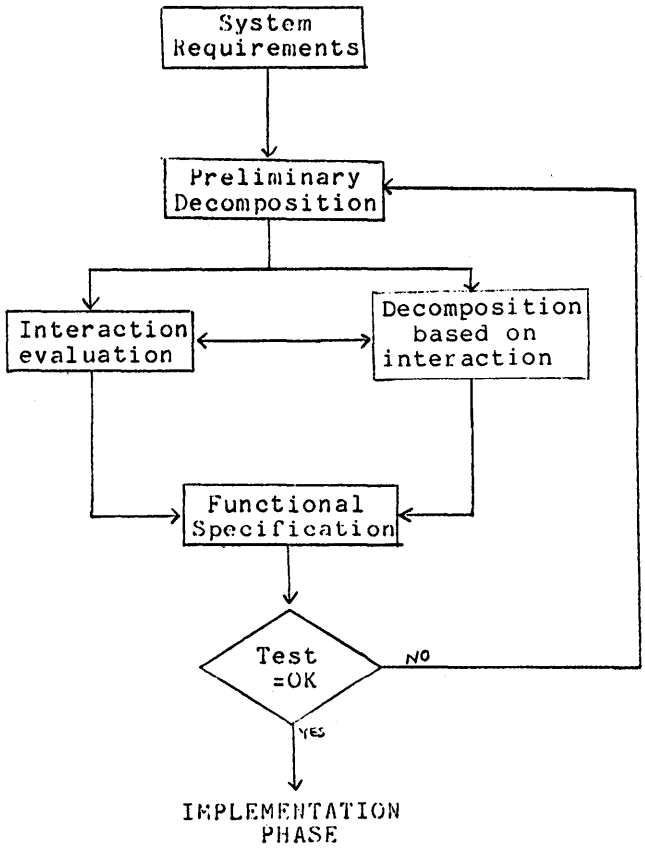


Figure 8—Design phase

Decomposition and partitioning

After the requirement process, a well defined, complete, consistent, unambiguous and testable set of system process specifications are produced. These system process specifications insure that the system behavior will be satisfactory to the customer and that the required system can plausibly be designed and implemented. Usually the specified system at this stage is so complex that it is very difficult if not impossible for the hardware designers to start implementing the system. In order for the design process to be manageable, it must be decomposed in such a way that most decisions can be made locally, based on data available within a local area of the developing system specifications.

The decomposition process can be accomplished by identifying the tightly coupled processes of the system and factoring the system into subsystems with respect to this criterion. The remaining steps of the development process can then independently (or nearly so) elaborate the design of each subsystem, maintaining the inter-subsystem interactions as design invariances. However, the decomposition must be carried out very carefully satisfying the following criteria:

(1) The decomposed system must be well-defined—they are consistent, complete, unambiguous and testable.
(2) The decomposed system must be capable of being integrated.

(3) The decomposed system must meet the resource allocation requirements, reliability requirements, performance requirements, etc.
(4) The decomposed system must satisfy the parametric logical specifications such that minor changes in the requirements would not require a redesign of the whole system.
(5) The decomposed system must be expandable so that future growth of the system can be easily incorporated.

In accomplishing the above criteria, a decomposition procedure will be used to guide the designer to generate a satisfactory decomposition of the system. The objective here is not to produce an optimal solution, but to develop a set of tools to help him to make his decisions. Most of the steps in the decomposition process will be automated to relieve the designer from tedious computations. However, some tradeoff decisions require human experience and interaction, and must be made by the designer. As a result, a close interaction of the designer in the decomposition process is required.

Preliminary decomposition

A large system will have many processes that are only loosely coupled, for example, the communication process and the data processing process. These loose subsets of tightly coupled processes can easily be identified by the designer and can be used to partition the system into subsystems. Thus, the whole system is decomposed into smaller subsystems, each representing some aspects of the original system from a different point of view. The number of such decomposed subsystems is application-dependent and sensitive to the identification of loosely coupled subsets of processes. By this preliminary decomposition, the system is decomposed into a structure according to its requirements and is ready for further analyses.

Decomposition based on interaction

The objective of interaction decomposition is to reduce the amount of interactions between subsystems. This in turn reduces the complexity of the interfaces and the amount of communication between subsystems. The interaction decomposition process can be divided into two cooperating steps: (i) decision process which requires direct interaction from the designer, and (ii) solution finding process which can be highly automated, Figure 9. In the decision process, the designer looks at the requirement specifications and assigns weights to the interaction between processes. These weights will be a function of communication cost, amount of traffic flow, degree of synchronization, ranking of the attributes, etc. By using this weighting function, tightly coupled processes will be assigned a high weight. Processes that should be in different modules (for example, due to reliability requirement) are assigned a very low weight. With

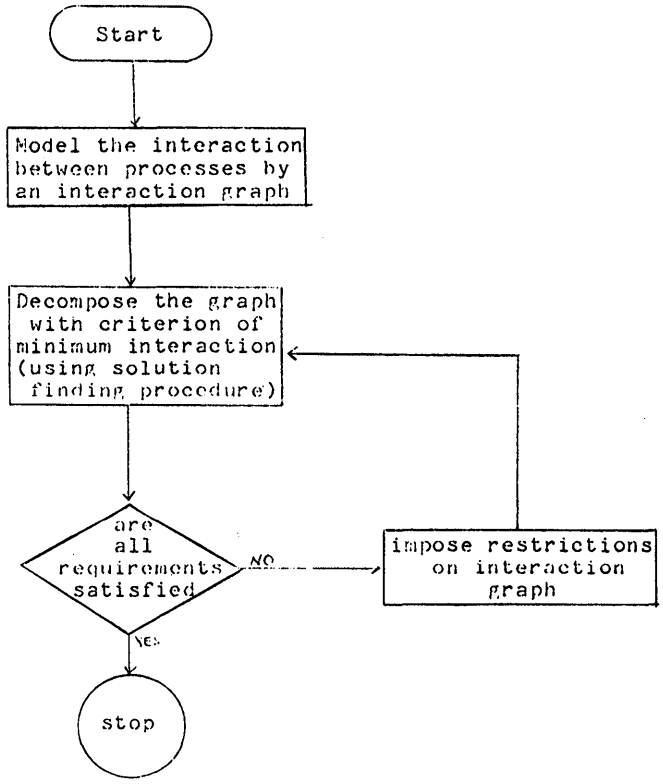


Figure 9—Decomposition based on interaction

this weight assigning function, we will then model the system by an interaction graph, Figure 10. In the interaction graph, nodes represent processes and the weight assigned to each arc corresponds to the amount of interaction of the two processes connected by the arc. In this way, the system can then be decomposed into subsystems automatically by the solution finding process. After the decomposition, the decomposed system is then checked by the designer to determine whether all the requirements are satisfied. The discrepancies are identified and the required changes are imposed onto the interaction graph which is analyzed by the solution finding step again. These two steps are iterated until all the specifications are met.

Solution finding procedure: In this step, the system which is represented by an interaction graph is partitioned into loosely coupled subsystems such that the system requirements are satisfied. First, the interaction cut-tree will be generated from the interaction graph by using the max. flow

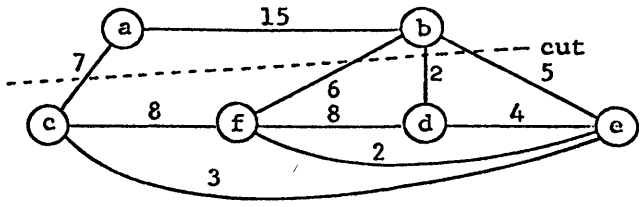


Figure 10—Interaction graph

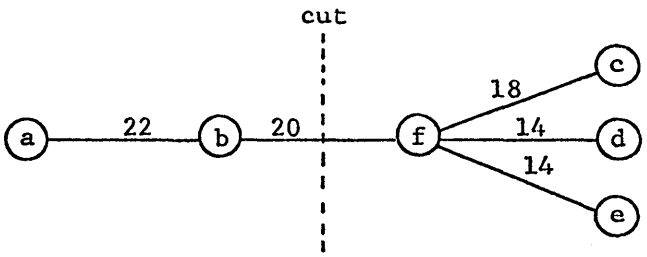


Figure 11—Interaction cut-tree

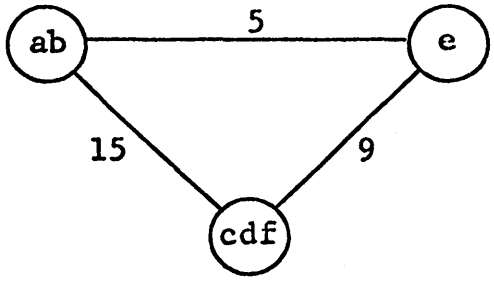


Figure 12—Decomposed system

min. cut algorithm.[19] It involves choosing two nodes arbitrarily in the interaction graph and finding the minimum cut set of the two nodes by the max. flow min. cut algorithm. This minimum cut divides the interaction graph into two subgraphs. Two other nodes are then chosen arbitrarily in one subgraph and the above procedure is repeated (with the other subgraphs being considered as macro nodes) until the interaction cut tree is generated. By this transformation, the interaction requirements between processes are represented very clearly by the interaction cut-tree, Figure 11. The minimal cut-set separating two nodes in the interaction graph is one to one corresponding to the minimal cut-set of the same two nodes in the interaction cut-tree. In addition to this, the values of the corresponding cut-sets in the two graphs are equal. For example, the minimal cut-set separating nodes a and f partitions the interaction graph and the interaction cut-tree identically into two subgraphs, {a,b} and {c,d,e,f}. The two cut sets both have values equal to 22. As a result, the decomposition can be performed on the interaction cut-tree rather than the interaction graph. This simplifies the computation greatly and displays clearly to the designer the condition of the system. The minimum cut can be identified easily in the cut-tree as it is the minimum weighted arc in the unique path connecting the two nodes.

After the interaction cut-tree has been generated, the system can be partitioned into loosely coupled subsystems such that the total weight of all the arcs in the cut-set is minimum. In addition to this, the partition should preserve the special configuration imposed by the requirements specified by the designer. For example, because of resource requirements, processes a and f, process a and d, and processes e and f have to be in different modules. In order for a and d to be on different modules, either arc (a,b), (b,f) or (f,d) has to be cut in the cut-tree. These requirements are then expressed in a table as shown in Table I.

The decomposition can now be achieved by finding the set of arcs in Table 1 such that each row in the table contains at least one cross in the arcs chosen. This can be solved by the set covering algorithm.[20] In our example, arcs (b,f) and (e,f) are chosen such that the interactions between subsystems are minimized, Figure 12. In general, this method will give a solution very close to the optimal solution and the computation complexity is very low when compared with that of generating the optimal solution.

TABLE I.—Constraint Table

arcs separating	(a,b)	(b,f)	(c,f)	(d,f)	(e,f)
a,f	X	X			
a,d	X	X		X	
e,f					X

Functional specification

The next major step in the design phase is functional specification of the partitioned processes. This functional specification is different from the process specification described in the requirement specification phase. The objective of the process specification is to define the interactions of the processes for the decomposition step. The objective of the functional specification here is to define the characteristics of the functions so as to enable optimization in the functions to processors mapping.

In the functional specification, all the processes in the same partition are considered as a single function. Similar to the process specification, the input and output relation, the precedence constraints and the interactions between different functions are determined. In addition to these, the characteristics of the function are defined. These include:

(i) types of operations to be performed—matrix operations, floating point or integer operations, etc.
(ii) resource requirements—storage requirement, processing power, etc.
(iii) speed requirements—frequency and execution speed of the function.

Based on these information and the processors available, the functions are mapped onto the available processors. In Reference 21, an efficient mapping algorithm of two processors system are discussed. For the general case of n processors, no efficient algorithm is known at this time and more research should be done in this area.

Verification of design

In order to be able to verify the correctness of the design and to evaluate the effectiveness of the control, the system must be modelled in some abstract model. This abstract

model should be amenable to analyze the intercommunication, the synchronization, the performance and the coordination of the functions. In this methodology, the Petri net model is chosen.

Petri nets display clearly the flow of information and control in systems, especially those which exhibit asynchronous and concurrent properties. However, in order to model the time constraints of real-time systems, the Petri net model is extended to include the execution time aspect of a system. In the following sections, the extended Petri net model and the analysis techniques will be discussed. In general, the designed system is modelled by a Petri net at all levels in the design phase. The Petri net is analyzed to guarantee the proper behavior of the system. For example, the Petri net models of the system before and after the decomposition are analyzed to ensure the consistency of decomposition.

Analyses of the system

Throughout the whole design phase, the system is modelled by a Petri net. By analyzing the liveness, the boundness and the proper termination properties of the Petri net,[22-25] the properties of the designed system can be unveiled. A Petri net is live if for each transition in the net, there always exists a firing sequence to fire it. By the liveness property of the Petri net, the designed system is guaranteed to be dead-lock free. A Petri net is bounded if for each place in the net, there exists an upper bound to the number of tokens that can be there simultaneously. By the boundness property of the Petri net, the number of buffers required between asynchronous processes can be determined and therefore information loss due to buffer overflow can be avoided. A Petri net is properly terminated if the Petri net always terminates in a well-defined manner. By the proper termination property, the designed system is guaranteed to function in a well behaved manner. (This point will be elaborated later.)

All the above three properties can be analyzed by constructing the reachability graph. (The method for constructing the reachability graph can be found in Reference 22.) In

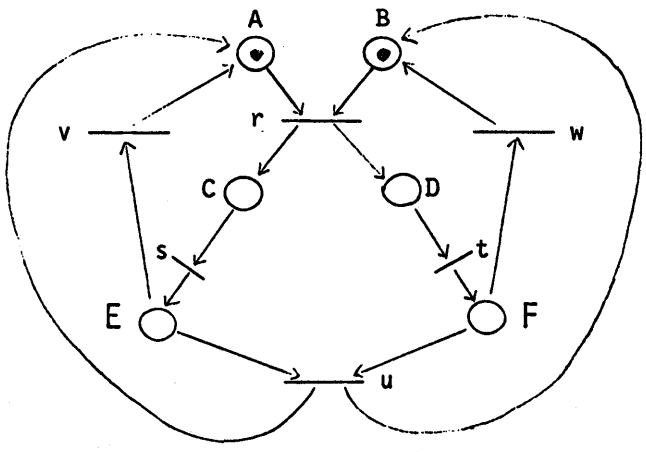


Figure 13—A live and bounded Petri net

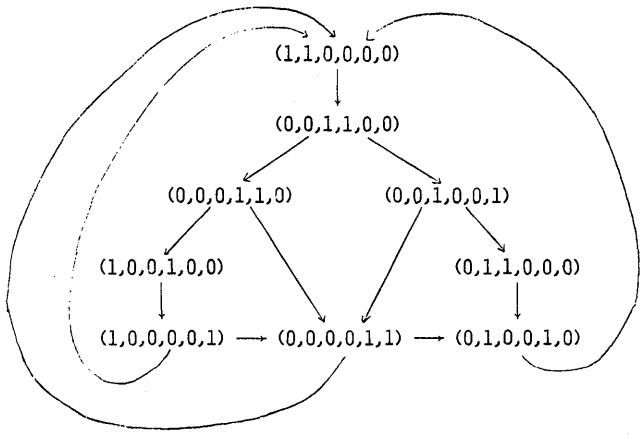


Figure 14—Reachability graph

Figure 13, the Petri net models a concurrent system with five processes: r, s, t, u and v. The reachability graph is shown in Figure 14. Since the reachability graph is strongly connected and all the entries are finite, the concurrent system is live and bounded. By these two properties, the concurrent system is guaranteed to be dead-lock free and to have enough buffers.

Extended Petri net

However in real world problems, especially in real-time systems, the execution time of a process is a very important aspect of a system. In order to model this aspect, the Petri net model is extended to include the notion of time. In the extended net, each transition is associated with two times, $t1$ and $t2$, where

$t1$=minimum time a transition can begin firing after being enabled (it can be zero)
and $t2$=maximum time a transition can remain not fired after being enabled ($t2>t1$)

Whenever a transition is enabled, it has to fire between times $t1$ and $t2$. In the case that a transition has $(t1,t2)$ as defined and an execution duration of u, it can be modelled by cascading two transitions with times $(t1,t2)$ and (u,u) associated to the transitions, Figure 15. By using this ap-

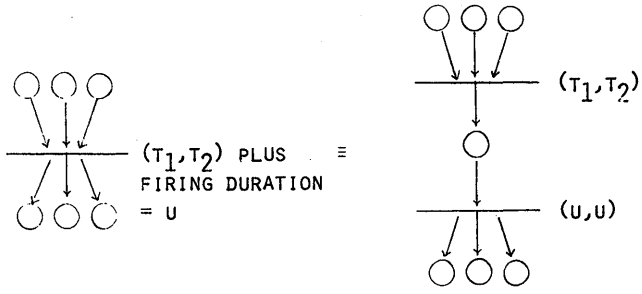


Figure 15—Timed Petri net

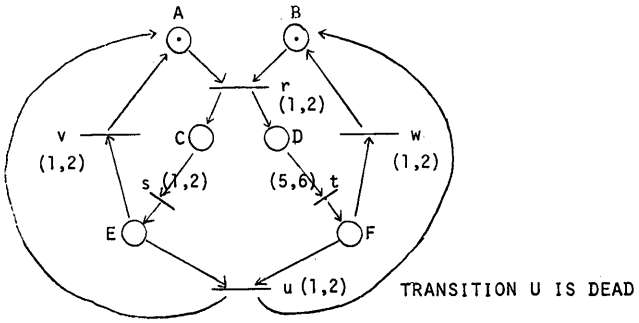


Figure 16—Non-live condition of a timed Petri net

proach, the problem of losing tokens during the firing of a transition is avoided.

By using the extended Petri net, the concurrent system discussed previously is analyzed again. The Petri net model of the concurrent system and its time constraints are shown in Figure 16. A careful study of the system will show that transition u is dead (it will never be fired). This is due to the long delay produced by transition t. After firing transition r, the token in place C will move into place E and then move into place A before the token from place D can move into place F. This situation is shown in the reachability graph in Figure 17. The two times associated to each arc is the earliest and the latest transition times of the arc. The two arcs t^1 and t^2 are blocked because their earliest transition time is longer than the latest transition time of the other transition emitting from the same node. Therefore by using the extended Petri net, the dead-lock situation due to time constraints can be detected. In addition to that, if the Petri net is a marked graph[26] and if the execution time, τ, of a transition is taken to be $t2$, the maximum computation rate, ρ, of each of the transition can be computed by[26]

$$\rho=\min\left\{\frac{N_k}{T_k} \mid k=1,2,\ldots n\right\}$$

where

$$T_k=\sum_{t_i\in C_k}\tau_i$$

is the sum of the times associated with transitions of circuit C_k and

$$N_k=\sum_{V_k\in C_k}M_k$$

is the number of tokens in places of circuit C.

In finding the dead-lock situation in the extended Petri net, the algorithm to find the earliest and latest transition times in the reachability graph is quite lengthy and complicated. For example, the arc t^2 is associated with times (3,5) rather than (5,6) because transition t is enabled before transition s is fired. This type of interdependency can be resolved by tracing back the reachability graph to find the time when the transition is enabled. The computation can be done by a computer. Although it may be very lengthy, it is worth the effort to guarantee the designed system to be dead-lock free.

Proper termination

A Petri net is properly terminated if a token is injected into the input place, the net produces a finite number of tokens and terminates with all the tokens in its output place.[23,24] This implies that if the corresponding system is initiated, it will always execute to completion without any intermediate results or pending conditions left in the system.

The proper termination property of a Petri net can be studied by its reachability graph. For example, the Petri net and its reachability graph are shown in Figure 18 and Figure 19. Since the only maximal node in the reachability graph is the state that place E contains a token, the Petri net is properly terminated.

The notion of proper termination can be used to verify the consistency of decomposition. For example, in Figure 20, transition s in global net N1 is decomposed into the subnet S. Virtual nodes A through G are then added to the subnet to form N2 which is then analyzed for proper termination.

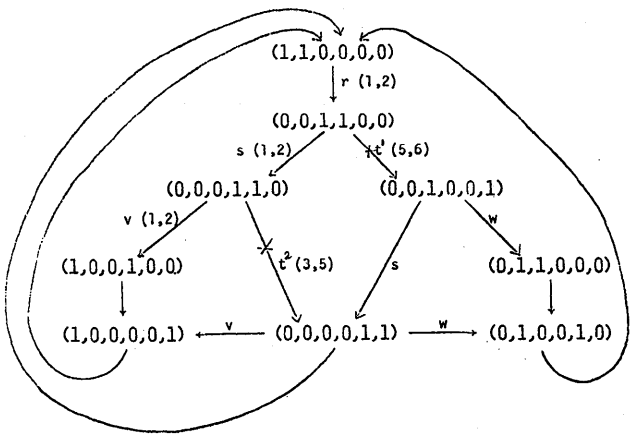


Figure 17—Reachability graph

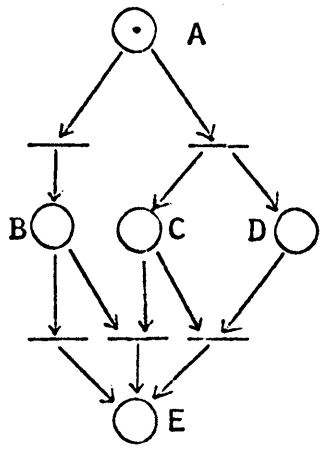


Figure 18—A properly terminated Petri net

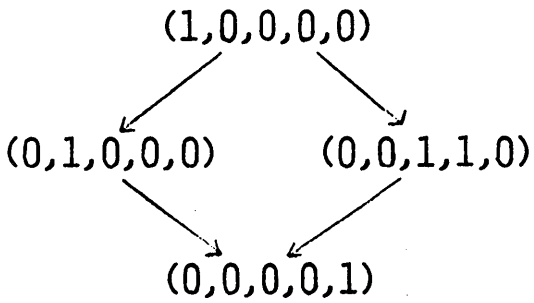


Figure 19—Reachability graph

If N2 is properly terminated, the decomposition is consistent. This guarantees that no data stay in the decomposed process and no extra data are generated when the decomposed process is substituted for the original process in the system. If time constraints, $(t1,t2)$, are added to each of the transitions, the time constraints of transition s can be verified by finding the minimum and maximum path lengths in the decomposed net.

The verification techniques discussed above are by no means complete. They are just some of the approaches for analyzing the synchronization and consistency of interacting

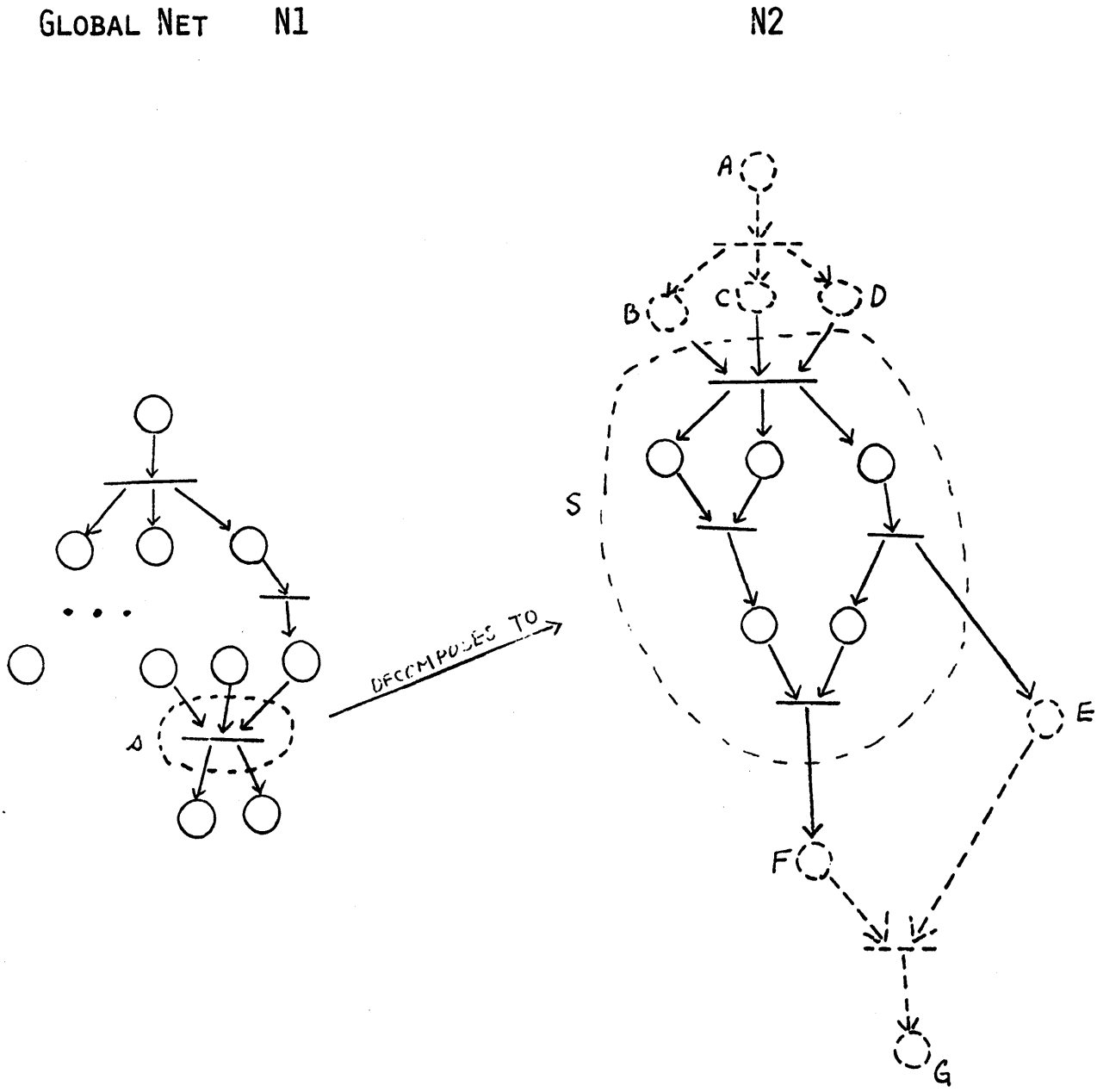


Figure 20—Consistency verification by Petri net

asynchronous processes. It is hoped that through these analysis techniques, synchronization errors in the designed system can be detected before implementation.

IMPLEMENTATION, EVALUATION AND VALIDATION

The implementation phase takes the virtual system and develops the system architecture. It then maps the system functions into either hardware or software functions. This step is greatly dependent on the technology, the architecture chosen, the physical constraints of the system. The final phase is the evaluation and validation of the system. This phase uses the bottom up validation approach. Both analytical modelling and simulation will be used. Because of the hierarchical decomposition, each subsystem to be analyzed should be small and therefore the complexity should be low. Mistakes or unfulfilled requirements found are traced back to the source of error. The system is then redesigned from that point.

CONCLUSION

This paper has discussed a systematic design procedure for a user oriented computer system. The main objective is to develop reliable, effective, modifiable systems with low cost and lead time. The methodology uses the concept of abstraction, stepwise refinement and modularity to design a DPS. By following the design guidelines of the methodology, the system can be developed systematically in a hierarchical manner.

The design methodology is divided into four successive phases: (i) requirement and specification phase, (ii) design phase, (iii) implementation phase, and (iv) evaluation and validation phase. The first two phases have been explored in detail in the previous sections. In the requirement and specification phase, the requirements and the characteristics of the system are elaborated and expressed in a formal specification. The basic steps involved, the primitives to be specified, and the choice of specification language are discussed. In the design phase, the system specifications are analyzed to generate the virtual system. A systematic decomposition procedure is proposed. Analysis and modelling techniques used in this phase are also discussed. By modelling the virtual system in an extended timed Petri net, the performance, the consistency and the liveness of the system can be determined. The last two phases of the methodology are only outlined briefly. They are too technology and architecture dependent and are beyond the scope of this paper.

ACKNOWLEDGMENTS

This research was supported by the Ballistic Missile Defense Advanced Technology Center (BMDATC) under contract DASG60-77-C-0138. We would also like to thank Mr. C. R. Vick, J. E. Scalf, D. Palmer and M. Mariani for many helpful discussions related to this work.

REFERENCES

1. Davis, C. G. and C. R. Vick, "The Software Development System," *Proceedings of the 2nd International Conference on Software Engineering,* 1976.
2. Belady, L. A. and M. M. Lehman, "The characteristics of Large Systems," IBM Research Report, RC 6785, Sept. 1977.
3. Lehman, M. M. and Parr, F. N., "Program Evolution and its impact on Software Engineering," Proceedings of the 2nd International Conference in Software Engineering, San Francisco, Oct. 1976.
4. Boehm, B. W. and R. K. McClean, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *Proceedings of the International Conference of Reliable Software,* 1975.
5. Bell, T. E. and T. A. Thayer, "Software Requirements: Are they Really a Problem?" *Proceedings of the 2nd International Conference on Software Engineering,* 1976.
6. Alford, P. C. and D. S. Taylor, "R-nets: A graph model for Real-time Software Requirements," *Proceedings of MRI Symposium on Computer Software Engineering,* 1976.
7. Long, A. B. et al., "A Methodology for the Development and Validation of Critical Software for Nuclear Power Plants," *Proceedings of the COMPSAC,* Nov. 1977.
8. Wymore, A. W., *System Engineering Methodology for Inter-Disciplinary Teams,* Wiley—Interscience, 1976.
9. Conn, A. P., "Specification of Reliable Large Scale Software Systems," Ph.D. Dissertation, Department of Electrical Engineering and Computer Sciences, U. of California, Berkeley, 1977.
10. Ross, D., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Trans. on Software Engineering,* Vol. SE-3, No. 1, Jan. 1977.
11. Teichroew, D. and Hershey, E. A. III, "PSA/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Trans. on Software Engineering,* Vol. SE-3, No. 1, Jan. 1977.
12. Hamilton, M. and S. Zeldin, "Higher Order Software Methodology for Defining Software," *IEEE Trans. on Software Engineering,* Vol. SE-2, No. 1, March 1976.
13. Rittel, H. W. Jr., "On the Planning Crises: System analysis of the First and Second Generations," *Bedriftsokonomen,* No. 8, pp. 390-396.
14. Gerard, D., *Theory of Value,* New York, Wiley 1959.
15. Fishburn, P. C., *Decision and Value Theory,* New York, Wiley 1964.
16. Mariani, M. P., "The Use of Payoff Trees in the Distributed Data Processing Design Process," Distributed Data Processing Technology FY 77 Research Conference Publications, 1977.
17. Ramamoorthy, C. V. and S. F. Ho, "Testing large software with Automated Software Evaluation Systems," *IEEE Trans. on Software Engineering,* March 1976.
18. Ramamoorthy, C. V. and T. Krishnarao, "The Design Issues in Distributed Computer Systems," Distributed Systems, Infotech State of the Art Report, 1976.
19. Ford, L. R. Jr. and D. R. Fulkerson, *Flow in Network,* Princeton University Press, Princeton, N.J., 1972.
20. Garfinkel, R. S. and G. L. Nemhauser, *Interger Programming,* Wiley—Interscience, 1972.
21. Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. on Software Engineering,* Vol. SE-3, No. 1, Jan. 1977.
22. Karp, R. M. and R. E. Miller, "Properties of a Model for Parallel Computation: Determinacy, Termination, Queueing," *SIAM J. Appl. Math.* 14, 6, Nov. 1966.
23. Gostelow, K. P., "Flow of Control, Resource Allocation and Proper Termination of Programs," Ph.D. Dissertation Computer Science Dept., U. of Calif., Los Angeles, Dec. 1971.
24. Postel, J. B., "A Graph Model Analysis of Computer Communications Protocols," Ph.D. Dissertation, Computer Science Dept., U. of Calif., Los Angeles, Jan. 1974.

25. Hack, M., "Decidability questions for Petri nets," Ph.D. Dissertation, Dept. of Electrical Engineering, M.I.T., Dec. 1975.
26. Ramchandani, C., "Analysis of Asynchronous Concurrent Systems by Petri Nets," Project MAC, TR-120, M.I.T., 1974.
27. Misunas, D., "Petri Net and Speed Independent Design," *Comm. ACM* 16, 8, Aug. 1973.
28. Peterson, J. L., "Petri Nets," *Computer Survey,* Vol. 9, No. 3, Sept. 1977.
29. Petri, C. A., "Communication with automata," Rome Air Develop. Cent., Suppl. 1 to Tech. Report No. RADC-TR-65-377. Reconnaissance—Intelligence Data Handling Branch, Rome Air Develop. Center, Griffin AFB, New York, Jan. 1966.
30. Hobbs, L. B., "The Role of Minicomputers and Microprocessers in Distributed Systems," *Distributed Systems,* Infotech State of the Art Report, 1976.
31. Holt, A. and F. Commoner, *Events and Conditions,* Applied Data Research, Wakefield, Mass., 1968.

VAX-11/780—A virtual address extension to the DEC PDP-11 family

by W. D. STRECKER
Digital Equipment Corporation
Maynard, Massachusetts

INTRODUCTION

Large virtual address space minicomputers

Perhaps the most useful definition of a minicomputer system is based on price: depending on one's perspective such systems are typically found in the $20K to $200K range. The twin forces of market pull—as customers build increasingly complex systems on minicomputers—and technology push—as the semiconductor industry provides increasingly lower cost logic and memory elements—have induced minicomputer manufacturers to produce systems of considerable performance and memory capacity. Such systems are typified by the DEC PDP-11/70. From an architectural point of view, the characteristic which most distinguishes many of these systems from larger mainframe computers is the size of the virtual address space: the immediately available address space seen by an individual process. For many purposes the 65K byte virtual address space typically provided on minicomputers (such as the PDP-11) has not been and probably will not continue to be a severe limitation. However, there are some applications whose programming is impractical in a 65K byte virtual address space, and perhaps most importantly, others whose programming is appreciably simplified by having a large virtual address space. Given the relative trends in hardware and software costs, the latter point alone will insure that large virtual address space minicomputers play an increasingly important role in minicomputer product offerings.

In principle, there is no great challenge in designing a large virtual address minicomputer system. For example, many of the large mainframe computers could serve as architectural models for such a system. The real challenge lies in two areas: compatibility—very tangible and important; and simplicity—intangible but nonetheless important.

The first area is preserving the customer's and the computer manufacturer's investment in existing systems. This investment exists at many levels: basic hardware (principally busses and peripherals); systems and applications software; files and data bases; and personnel familiar with the programming, use, and operation of the systems. For example, just recently a major computer manufacturer abandoned a major effort for new computer architectures in favor of evolving its current architectures.[1]

The second intangible area is the preservation of those attributes (other than price) which make minicomputer systems attractive. These include approachability, understandability, and ease of use. Preservation of these attributes suggests that simply modelling an extended virtual address minicomputer after a large mainframe computer is not wholly appropriate. It also suggests that during architectural design, tradeoffs must be made between more than just performance, functionality, and cost. Performance or functionality features which are so complex that they appreciably compromise understanding or ease of use must be rejected as inappropriate for minicomputer systems.

VAX-11 overview

VAX-11 is the Virtual Address eXtention of PDP-11 architecture.[2,3] The most distinctive feature of VAX-11 is the extension of the virtual address from 16 bits as provided on the PDP-11 to 32 bits. With the 8-bit byte the basic addressable unit, the extension provides a virtual address space of about 4.3 gigabytes which, even given rapid improvement in memory technology, should be adequate far into the future.

Since maximal PDP-11 compatibility was a strong goal, early VAX-11 design efforts focused on literally extending the PDP-11: preserving the existing instruction formats and instruction set and fitting the virtual address extension around them. The objective here was to permit, to the extent possible, the running of existing programs in the extended virtual address environment. While realizing this objective was possible (there were three distinct designs), it was felt that the extended architecture designs were overly compromised in the areas of efficiency, functionality, and programming ease.

Consequently, it was decided to drop the constraint of the PDP-11 instruction format in designing the extended virtual address space or *native mode* of the VAX-11 architecture. However, in order to run existing PDP-11 programs, VAX-11 includes a PDP-11 *compatibility mode*. Compatibility

mode provides the basic PDP-11 instruction set less only privileged instructions (such as HALT) and floating point instructions (which are optional on most PDP-11 processors and not required by most PDP-11 software).

In addition to compatibility mode, a number of other features to preserve PDP-11 investment have been provided in the VAX-11 architecture, the VAX-11 operating system VAX/VMS, and the VAX-11/780 implementation of the VAX-11 architecture. These features include:

1. The equivalent native mode data types and formats are identical to those on the PDP-11. Also, while extended, the VAX-11 native mode instruction set and addressing modes are very close to those on the PDP-11. As a consequence VAX-11 native mode assembly language programming is quite similar to PDP-11 assembly language programming.
2. The VAX-11/780 uses the same peripheral busses (Unibus and Massbus) as the PDP-11 and uses the same peripherals.
3. The VAX/VMS operating system is an evolution of the PDP-11 RSX-11M and IAS operating systems, offers a similar although extended set of system services, and uses the same command languages. Additionally, VAX/VMS supports most of the RSX-11M/IAS system service requests issued by programs executing in compatibility mode.
4. The VAX/VMS file system is the same as used on the RSX-11M/IAS operating systems permitting interchange of files and volumes. The file access methods as implemented by the RMS record manager are also the same.
5. VAX-11 high level language compilers accept the same source languages as the equivalent PDP-11 compilers and execution of compiled programs gives the same results.

The coverage of all these aspects of VAX-11 is well beyond the scope of any single paper. The remainder of this paper discusses the design of the VAX-11 native mode architecture and gives an overview of the VAX-11/780 system.

VAX-11 NATIVE ARCHITECTURE

Processor state

Like the PDP-11, VAX-11 is organized around a general register processor state. This organization was favored because access to operands stored in general registers is fast (since the registers are internal to the processor and register accesses do not need to pass through a memory management mechanism) and because only a small number of bits in an instruction are needed to designate a register. Perhaps most importantly, the registers are used (as on the PDP-11) in conjunction with a large set of addressing modes which permit unusually flexible operand addressing methods.

Some consideration was given to a pure stack based architecture. However it was rejected because real program data suggests the superiority of two or three operand instruction formats.[4] Actually VAX-11 is quite stack oriented, and although it is not optimally encoded for the purpose, can easily be used as a pure stack architecture if desired.

VAX-11 has 16 32-bit general registers (denoted R0-R15) which are used for both fixed and floating point operands. This is in contrast to the PDP-11 which has eight 16-bit general registers and six 64-bit floating point registers. The merged set of fixed and floating registers were preferred because it simplifies programming and permits a more effective allocation of the registers.

Four of the registers are assigned special meaning in the VAX-11 architecture:

1. R15 is the *program counter* (PC) which contains the address of the next byte to be interpreted in the instruction stream.
2. R14 is the *stack pointer* (SP) which contains the address of the top of the processor defined stack used for procedure and interrupt linkage.
3. R13 is the *frame pointer* (FP). The VAX-11 procedure calling convention builds a data structure on the stack called a stack frame. FP contains the address of this structure.
4. R12 is the *argument pointer* (AP). The VAX-11 procedure calling convention uses a data structure called an argument list. AP contains the address of this structure.

The remaining element of the user visible processor state (additional processor state seen mainly by privileged procedures is discussed later) is the 16-bit *processor status word* (PSW). The PSW contains the N, Z, V, and C condition codes which indicate respectively whether a previous instruction had a negative result, a zero result, a result which overflowed, or a result which produced a carry (or borrow). Also in the PSW are the IV, DV, and FU bits which enable processor trapping on integer overflow, decimal overflow, and floating underflow conditions respectively. (The trapping on conditions of floating overflow and divide by zero for any data type are always enabled.)

Finally, the PSW contains the T bit which when set forces a trap at the end of each instruction. This trap is useful for program debugging and analysis purposes.

Data types and formats

The VAX-11 data types are a superset of the PDP-11 data types. Where the PDP-11 and VAX-11 have equivalent data types the formats (representation in memory) are identical. Data type and data format identity is one of the most compelling forms of compatibility. It permits free interchange of binary data between PDP-11 and VAX-11 programs. It facilitates source level compatibility between equivalent PDP-11 and VAX-11 languages. It also greatly facilitates hardware implementation of and software support of the PDP-11 compatibility mode in the VAX-11 architecture.

The VAX-11 data types divide into five classes:

1. Integer data types are the 8-bit *byte*, the 16-bit *word*, the 32-bit *longword*, and the 64-bit *quadword*. Usually these data types are considered signed with negative values represented in two's complement form. However, for most purposes they can be interpreted as unsigned and the VAX-11 instruction set provides support for this interpretation.
2. Floating data types are the 32-bit *floating* and the 64-bit *double* floating. These data types are binary normalized, have an 8-bit signed exponent, and have a 25- or 57-bit signed fraction with the redundant most significant fraction bit not represented.
3. The *variable bit field* data type is 0 to 32 bits located arbitrarily with respect to addressable byte boundaries. A bit field is specified by three operands: the address of a byte, the starting bit position P with respect to bit 0 of that byte, and the size S of the field. The VAX-11 instruction set provides for interpreting the field as signed or unsigned.
4. The *character string* data type is 0 to 65535 contiguous bytes. It is specified by two operands: the length and starting address of the string. Although the data type is named "character string", no special interpretation is placed on the values of the bytes in the character string.
5. The *decimal string* data types are 0 to 31 digits. They are specified by two operands: a length (in digits) and a starting address. The primary data type is *packed decimal* with two digits stored in each byte except that the byte containing the least significant digit contains a single digit and the sign. Two ASCII character decimal types are supported: *leading separate* sign and *trailing embedded* sign. The leading separate type is a "+", "−", or "⟨blank⟩" (equivalent to "+") ASCII character followed by 0 to 31 ASCII decimal digit characters. A trailing embedded sign decimal string is 0 to 31 bytes which are ASCII decimal digit characters except for the character containing least significant digit which is an arbitrary encoding of the digit and sign.

All of the data types except field may be stored on arbitrary byte boundaries—there are no alignment constraints. The field data type, of course, can start on an arbitrary bit boundary.

Attributes of and symbolic representations for most of the data types are given in Table I and Figure 1.

TABLE I.—Data Types

DATA TYPE	SIZE	RANGE (decimal)	
Integer		Signed	Unsigned
Byte	8 bits	-128 to +127	0 to 255
Word	16 bits	-32768 to +32767	0 to 65535
Longword	32 bits	-2^{31} to $+2^{31}-1$	0 to $2^{32}-1$
Quadword	64 bits	-2^{63} to $+2^{63}-1$	0 to $2^{64}-1$
Floating Point		$\pm 2.9 \times 10^{-37}$ to 1.7×10^{38}	
Floating	32 bits	approximately seven decimal digits precision	
Double Floating	64 bits	approximately sixteen decimal digits precision	
Packed Decimal String	0 to 16 bytes (31 digits)	numeric, two digits per byte sign in low half of last byte	
Character String	0 to 65535 bytes	one character per byte	
Variable-length Bit Field	0 to 32 bits	dependent on intrepretation	

Instruction format and address modes

Most architectures provide a small number of relatively fixed instruction formats. Two problems often result. First, not all operands of an instruction have the same specification generality. For example, one operand must come from memory and another from a register; or one must come from the stack and another from memory. Second, only a limited number of operands can be accommodated: typically one or two. For instructions which inherently require more operands (such as field or string instructions), the additional operands are specified in ad hoc ways: small literal fields in instructions, specific registers or stack positions, or packed in fields of a single operand. Both these problems lead to increased programming complexity: they require superfluous move type instructions to get operands to places where they can be used and increase competition for potentially scarce resources such as registers.

To avoid these problems two criteria were used in the design of the VAX-11 instruction format: (1) all instructions should have the "natural" number of operands and (2) all operands should have the same generality in specification. These criteria led to a highly variable instruction format. An instruction consists of a one or two* byte *opcode* followed by the specifications for n operands ($n \geq 0$) where n is an implicit property of the opcode. An operand specification is one to 10 bytes in length and consists of a one or two byte *operand specifier* followed by (as required) zero to eight

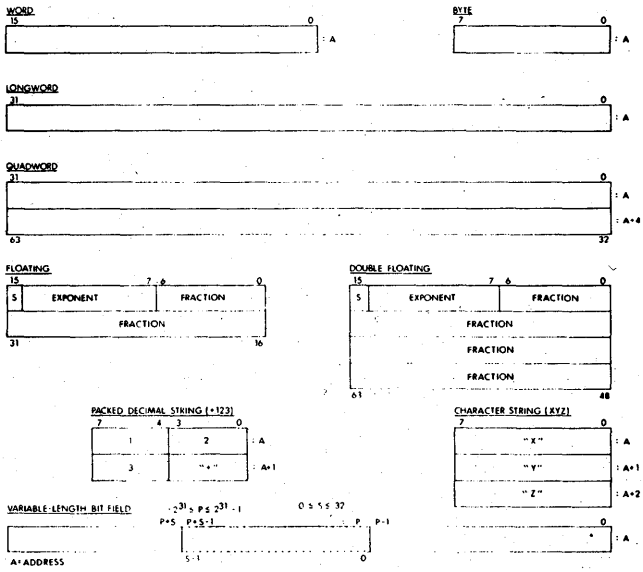


Figure 1—Data formats

* No currently defined instructions use two byte opcodes.

bytes of *specifier extension*. The operand specifier includes the address mode and designation of any registers needed to locate the operand. A specifier extension consists of a displacement, an address, or immediate data.

The VAX-11 address modes are with one exception a superset of the PDP-11 address modes. The PDP-11 address mode autodecrement deferred was omitted from VAX-11 because it was rarely used.

Most operand specifiers are one byte long and contain two 4-bit fields: the high order field (bits 7:4) contains the address mode designator and the lower field (bits 3:0) contains a general register designator. The address modes include:

1. *Register mode* in which the designated register contains the operand.
2. *Register deferred mode* in which the designated register contains the address of the operand.
3. *Autodecrement mode* in which the contents of the designated register are first decremented by the size (in bytes) of the operand and then used as the address of the operand.
4. *Autoincrement mode* in which the contents of the designated register are first used as the address of the operand and are then incremented by the size of the operand. Note that if the designated register is PC, the operand is located in the instruction stream. This use of autoincrement mode is called *immediate* mode. In immediate mode the one to eight bytes of data are the specifier extension.

   Autoincrement mode can be used sequentially to process a vector in one direction and autodecrement mode used to process a vector in the opposite direction. Autoincrement, register deferred, and autodecrement modes can be applied to a single register to implement a stack data structure: autodecrement to "push", autoincrement to "pop", and register deferred to access the top of the stack.
5. *Autoincrement deferred mode* in which the contents of the designated register are used as the address of a longword in memory which contains the address of the operand. After this use, the contents of the register are incremented by four (the size in bytes of the longword address). Note that if PC is the designated register, the absolute address of the operand is located in the instruction stream. This use of autoincrement deferred mode is termed *absolute* mode. In absolute mode the 4-byte address is the specifier extension.
6. *Displacement mode* in which a displacement is added to the contents of the designated register to form the operand address. There are three displacement modes depending on whether a signed byte, word, or longword displacement is the specifier extension. These modes are termed byte, word, and longword displacement respectively. Note that if PC is the designated register, the operand is located relative to PC. For this use the modes are termed byte, word, and longword *relative* mode respectively.
7. *Displacement deferred mode* in which a displacement is added to the designated register to form the address of a longword containing the address of the operand. There are three displacement deferred modes depending on whether a signed byte, word, or longword displacement is the specifier extension. These modes are termed byte, word, and longword displacement respectively. Note that if PC is the designated register, the operand address is located relative to PC. For this use the modes are termed byte, word, and longword *relative deferred* mode respectively.
8. *Literal mode* in which the operand specifier itself contains a 6-bit literal which is the operand. For integer data types the literal encodes the values 0-63; for floating data types the literal includes three exponent and three fraction bits to give 64 common values.
9. *Index mode* which is not really a mode but rather a one byte prefix operator for any other mode which evaluates to a memory address (i.e., all modes except register and literal). The index mode prefix is cascaded with the operand specifier for that mode (called the base operand specifier) to form an aggregate two byte operand specifier. The base operand specifier is used in the normal way to evaluate a base address. A copy of the contents of the register designated in the index prefix is multiplied by the size (in bytes) of the operand and added to the base address. The sum is the final operand address. There are three advantages to the VAX-11 form of indexing: (a) the index is scaled by the data size and thus the index register maintains a logical rather than a byte offset into an indexed data structure, (b) indexing can be applied to any of the address modes which generate memory addresses and this results in a comprehensive set of indexed addressing methods, and (c) the space required to specify indexing and the index register is paid only when indexing is used.

The VAX-11 assembler syntax for the address modes is given in Figure 2. The bracketed ({ }) notation is optional

Literal (Immediate)	{S↑ I↑} #constant	
Register	Rn	
Register Deferred	(Rn)	Indexed [Rx]
Autodecrement	-(Rn)	
Autoincrement	(Rn)+	
Autoincrement Deferred	@(Rn)+	
(Absolute)	@#address	
Displacement	{B↑ W↑ L↑} displacement(Rn) address	
Displacement Deferred	@{B↑ W↑ L↑} displacement(Rn) address	

n = 0 through 15
x = 0 through 14

Figure 2—Assembler syntax

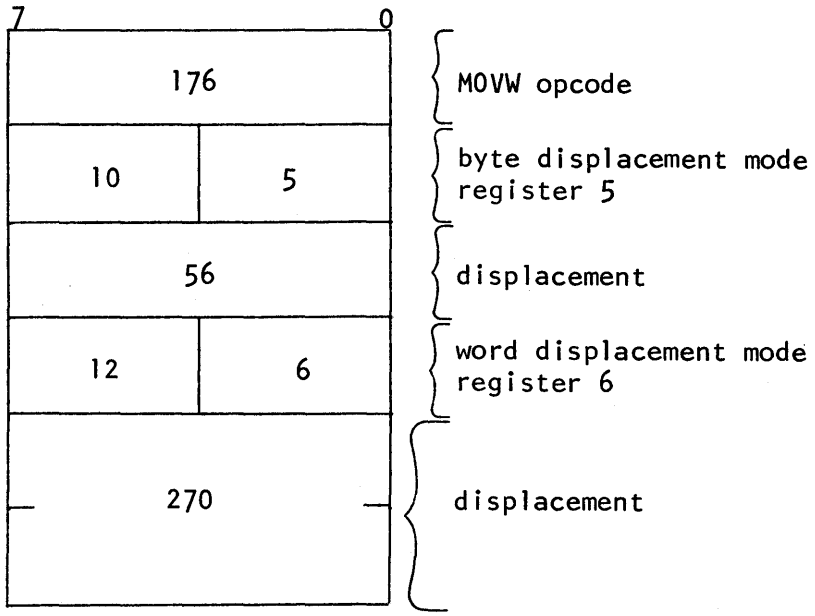


Figure 3—MOVW 56 (R5), 270(R7)

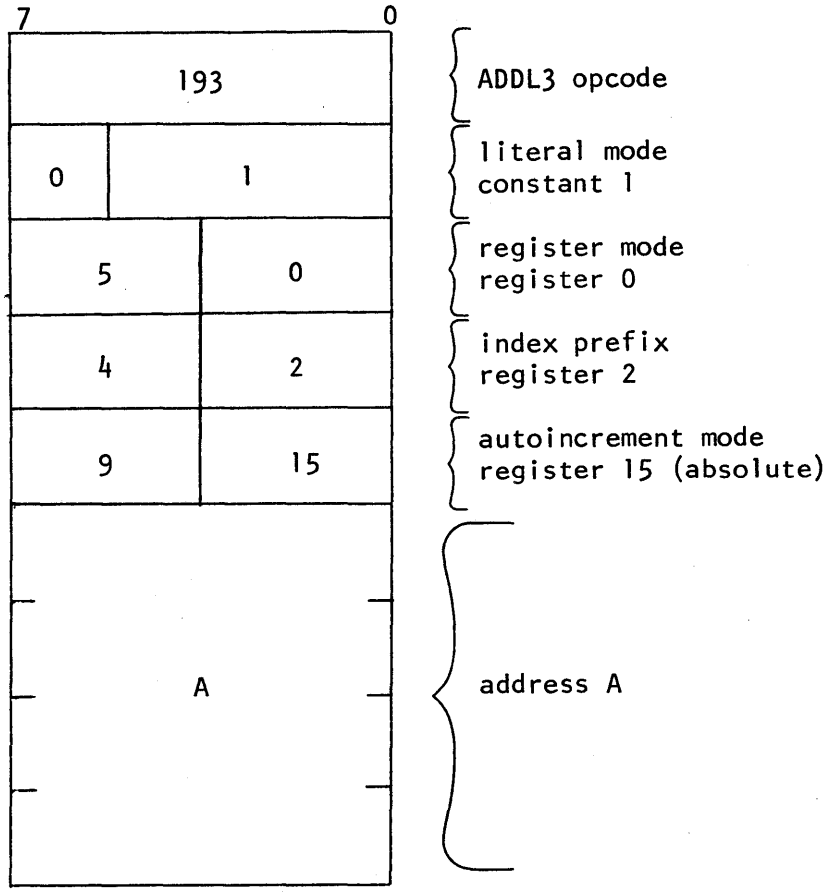


Figure 4—ADDL3 #1, RO, @#A[R2]

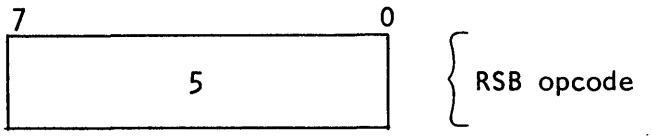


Figure 5—RSB

and the programmer rarely needs to be concerned with displacement sizes or whether to choose literal or immediate mode. The programmer writes the simple form and assembler chooses the address mode which produces the shortest instruction length.

In order to give a better feeling for the instruction format and assembler notation, several examples are given in Figures 3-5. In Figure 3 is an instruction which moves a word from an address which is 56 plus the contents of R5 to an address which is 270 plus the contents of R6. Note, that the displacement 56 is representable in a byte while the displacement 270 requires a word. The instruction occupies 6 bytes. In Figure 4 is an instruction which adds 1 to a longword in R0 and stores the result at a memory address which is the sum of A and 4 times the contents of R2. This instruction occupies 9 bytes. Finally, in Figure 5 is a return from subroutine instruction. It has no explicit operands and occupies a single byte.

The only significant instance where there is non-general specification of operands is in the specification of targets for branch instructions. Since invariably the target of a branch instruction is a small displacement from the current PC, most branch instructions simply take a one byte PC relative displacement. This is exactly as if byte displacement mode were used with the PC used as the register, except that the operand specifier byte is not needed. Because of the pervasiveness of branch instructions in code, this one byte saving results in a non-trivial reduction in code size. An example of the branch instruction branch on equal is given in Figure 6.

Instruction set

A major goal of the VAX-11 instruction set design was to provide for effective compiler generated code. Four decisions helped to realize this goal:

1. A very regular and consistent treatment of operators. Thus, for example, since there is a divide longword instruction, there are also divide word and divide byte instructions.

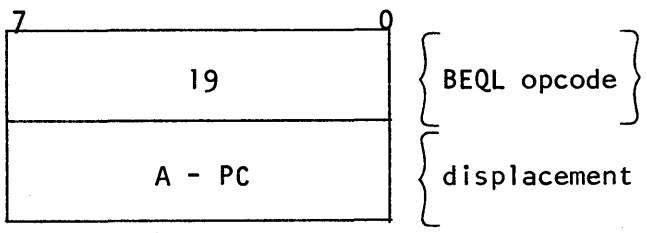


Figure 6—BEQL A

2. An avoidance of instructions unlikely to be generated by a compiler.
3. Inclusion of several forms of common operators. For example the integer add instructions are included in three forms: (a) one operand where the value one is added to an operand, (b) two operands where one operand is added to a second, and (c) three operands where one operand is added to a second and the result stored in a third. Since the VAX-11 instruction format allows fully general specifications of the operands, VAX-11 programs often have the structure (though not the encoding) of the canonic program form proposed in Reference 5.
4. Replacement of common instruction sequences with single instructions. Examples of this include procedure calling, multiway branching, loop control, and array subscript calculation.

The effect of these decisions is reflected in several observations. First, despite the larger virtual address and instruction set support for more data types, compiler (and hand) generated code for VAX-11 is typically smaller than the equivalent PDP-11 code for algorithms operating on data types supported by the PDP-11. Second, of the 243 instructions in the instruction set about 75 percent are generated by the VAX-11 FORTRAN compiler. Of the instructions not generated, most operate on data types not part of the FORTRAN language.

A complete list of the VAX-11 instructions is given in the Appendix. The following gives an overview of the instruction set.

1. *Integer logic and arithmetic*—Byte, word, and longword are the primary data types. A fairly conventional group of arithmetic and logical instructions is provided. The result generating dyadic arithmetic and logical instructions are provided in two and three operand forms. A number of optimizations are included: clear which is a move of zero; test which is a compare against zero; and increment and decrement which are an optimization of add one and subtract one respectively. A complete set of converts is provided which covers both the integer and the floating data types. In contrast to other architectures only a few shift type instructions are provided: it was felt that shifts are mostly used for field isolation which is much more conveniently done with the field instructions described later. In order to support greater than longword precision integer operations, a few special instructions are provided: extended multiply and divide and add with carry and subtract with carry.
2. *Floating point instructions*—Again a conventional group of instructions are included with result producing dyadic operators in two and three operand forms. Several specialized floating point instructions are included. The extended modulus instruction multiplies two floating operands together and stores the integer and fraction parts of the product in separate result operands. The polynomial instruction computes a polynomial

from a table of coefficients in memory. Both these instructions employ greater than normal precision and are useful in high accuracy mathematical routines. A convert rounded instruction is provided which implements the ALGOL rather than FORTRAN conventions for converting from floating point to integer.

3. *Address instructions*—The move address instructions store in the result operand the effective address of the source operand. The push address optimizations push on the stack (defined by SP) the effective address of the source operand. The latter are used extensively in subroutine calling.
4. *Field instructions*—The extract field instructions extract a 0 to 32-bit field, sign- or zero-extend it if it is less than 32 bits, and store it in a longword operand. The compare field instructions compare a (sign- or zero-extended if necessary) field against a longword operand. The find first instructions find the first occurrence of a set or clear bit in a field.
5. *Control instructions*—There is a complete set of conditional branches supporting both a signed and, where appropriate, an unsigned interpretation of the various data types. These branches test the condition codes and take a one byte PC relative branch displacement. There are three unconditional branch instructions: the first taking a one byte PC relative displacement, the second taking a word PC relative displacement, and the third—called jump—taking a general operand specification. Paralleling these three instructions are three branch to subroutine instructions. These push the current PC on the stack before transferring control. The single byte return from subroutine instruction returns from subroutines called by these instructions. There is a set of branch on bit instructions which branch on the state of a single bit and, depending on the instruction, set, clear, or leave unchanged that bit.

   The add compare and branch instructions are used for loop control. A step operand is added to the loop control operand and the sum compared against a limit operand. The result of the comparison determines whether the branch is taken. The sense of the comparison is based on the sign of the step operand. Optimizations of loop control include the add one and branch instructions which assume a step of one and the subtract one and branch instructions which assume a step of minus one and a limit of zero.

   The case instructions implement the computed go to in FORTRAN and case statements in other languages. A selector operand is checked to see that it lies in range and is then used to select one of table of PC relative branch displacements following the instruction.
6. *Queue instructions*—The queue representation is a doubly linked circular list. Instructions are provided to insert an item into a queue or to remove an item from a queue.
7. *Character string instructions*—The general move character instruction takes five operands specifying the lengths and starting addresses of the source and destination strings and a fill character to be used if the source string is shorter than the destination string. The instruction functions correctly regardless of string overlap. An optimized move character instruction assumes the string lengths are equal and takes three operands. Paralleling the move instructions are two compare character instructions. The move translated characters instruction is similar to the general move character instruction except that the source string bytes are translated by a translation table specified by the instruction before being moved to destination string. The move translated until escape instruction stops if the result of a translation matches the escape character specified by one of its operands. The locate and skip character instructions find respectively the first occurence or non-occurence of a character in a string. The scan and span instructions find respectively the first occurence or non-occurence of a character within a specified character set in a string. The match characters instruction finds the first occurence of a substring within a string which matches a specified pattern string.
8. *Packed decimal instructions*—A conventional set of arithmetic instructions is provided. The arithmetic shift and round instruction provides decimal point scaling and rounding. Converts are provided to and from longword integers, leading separate decimal strings, and trailing embedded decimal strings. A comprehensive edit instruction is included.

VAX-11 procedure instructions

A major goal of the VAX-11 design was to have a single system wide procedure calling convention which would apply to all inter-module calls in the various languages, calls for operating system services, and calls to the common run time system. Three VAX-11 instructions support this convention: two call instructions which are indistinguishable as far as the called procedure is concerned and a return instruction.

The call instructions assume that the first word of a procedure is an *entry mask* which specifies which registers are to be used by the procedure and thus need to be saved. (Actually only R0-R11 are controlled by the entry mask and bits 15:12 of the mask are reserved for other purposes.) After pushing the registers to be saved on the stack, the call instruction pushes AP, FP, PC, a longword containing the PSW and the entry mask, and a zero valued longword which is the initial value of a condition handler address. The call instruction then loads FP with the contents of SP and AP with the argument list address. The appearance of the stack frame after the call is shown in the upper part of Figure 7.

The form of the argument list is shown in the lower part of Figure 7. It consists of an argument count and list of longword arguments which are typically addresses. The CALLG instruction takes two operands: one specifying the procedure address and the other specifying the address of the argument list assumed arbitrarily located in memory.

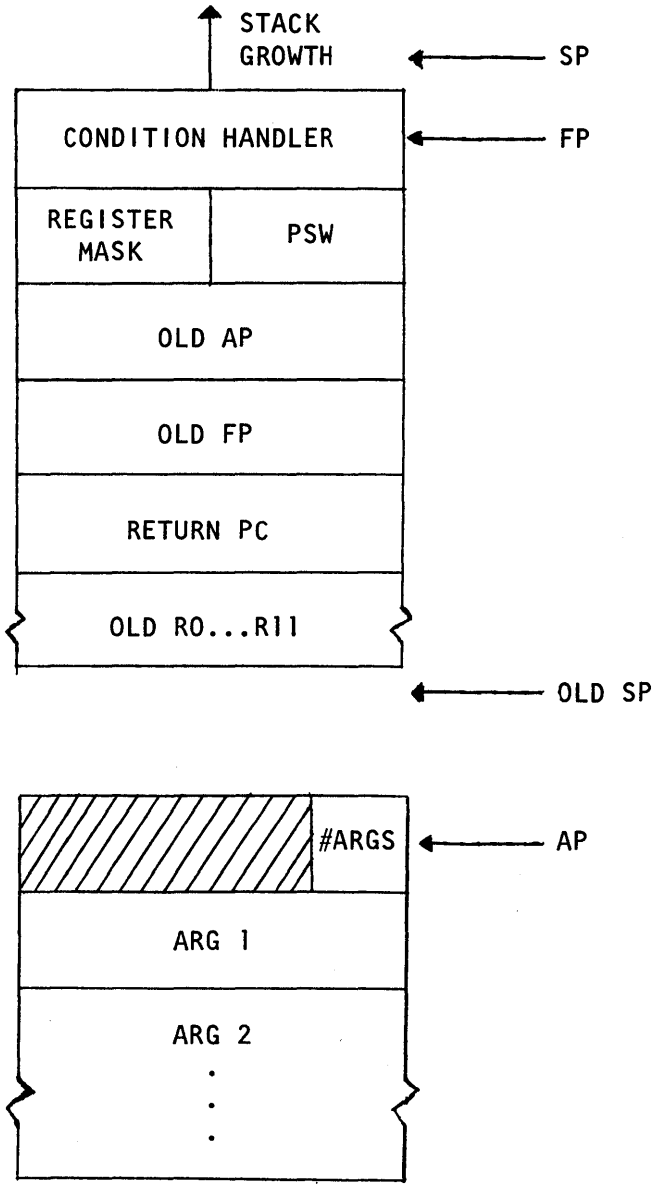


Figure 7—Stack frame

The CALLS instruction also takes two operands: one the procedure address and the other an argument count. CALLS assumes that the arguments have been pushed on the stack and pushes the argument count immediately prior to saving the registers controlled by the entry mask. It also sets bit 13 of the saved entry mask to indicate a CALLS instruction was used to make the call.

The return instruction uses FP to locate the stack frame. It loads SP with the contents of FP and restores PSW through PC by popping the stack. The saved entry mask controls the popping and restoring of R11 through R0. Finally if the bit indicating CALLS was set, the argument list is removed from the stack.

Memory management design alternatives

Memory management comprises the mechanisms used (1) to map the virtual addresses generated by processes to physical memory addresses, (2) to control access to memory (i.e., to control whether a process has read, write, or no access to various areas of memory), and (3) to allow a process to execute even if all of its virtual address space is not simultaneously mapped to physical memory (i.e., to provide so called virtual memory facilities). The memory management proved to be the most difficult part of the architecture to design. Three alternatives were pursued and full designs were completed for the first two alternatives and nearly completed for the third. The three alternatives* were:

1. A paged form of memory management with access control at the page level and a small number (4) of hierarchical access modes whose use would be dedicated to specific purposes. This represented an evolution of the PDP-11/70 memory management.
2. A paged and segmented form with access control at the segment level and a larger number (8) of hierarchical access modes which would be used quite generally. Although it differed in a number of ways, the design was motivated by the Multics[6,7] architecture and the Honeywell 6180 implementation.
3. A capabilities[8,9] form with access control provided by the capabilities and the ability to page larger objects described by the capabilities.

The first alternative was finally selected. The second alternative was rejected because it was felt that the real increase in functionality provided inadequately offset the increased architectural complexity. The third alternative appeared to offer functionality advantages that could be useful over the longer term. However, it was unlikely that these advantages could be exploited in the near term. Further it appeared that the complexity of the capabilities design was inappropriate for a minicomputer system.

Memory mapping

The 4.3 gigabyte virtual address space is divided into four regions as shown in Figure 8. The first two regions—the *program* and *control* regions—comprise the per process virtual address space which is uniquely mapped for each process. The second two regions—the *system* region and a region reserved for future use—comprise the system virtual address space which is singly mapped for all processes.

Each of the regions serves different purposes. The program region contains user programs and data and the top of the region is a dynamic memory allocation point. The control

* It should not be construed that memory management is independent of the rest of the architecture. The various memory management alternatives required different definitions of the addressing modes and different instruction level support for addressing.

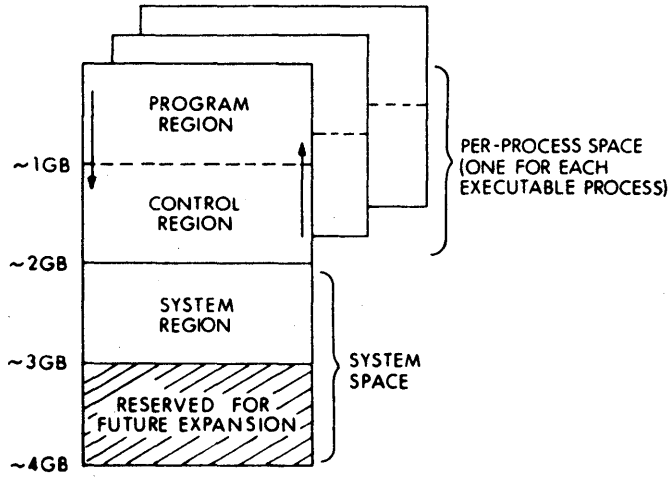


Figure 8—Virtual address space

region contains operating system data structures specific to the process and the user stack. The system region contains procedures which are common to all processes (such as those that comprise the operating system and RMS) and (as will be seen later) page tables.

A virtual address has the structure shown in the upper part of Figure 9. Bits 8:0 specify a byte within a 512 byte *page* which is the basic unit of mapping. Bits 29:9 specify a *virtual page number* (VPN). Bits 31:30 select the virtual address region. The mechanism of mapping consists of using the region select bits to select a *page table* which consists of *page table entries* (PTEs). After a check that it is not too large, the VPN is used to index into the page table to select a PTE. The PTE contains either (1) 21-bit physical *page frame number* which is concatenated with the nine low order byte in page bits to form a 30-bit physical address shown in the lower part of Figure 9, or (2) an indication that the virtual page accessed is not in physical memory. The latter case is called a *page fault*. Instruction execution in the current procedure is suspended and control is transferred to an operating system procedure which will cause the missing virtual page to be brought into physical memory. At this point instruction execution in the suspended procedure can resume transparently.

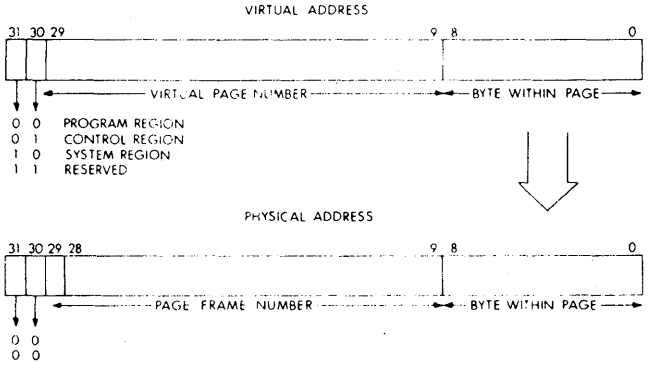


Figure 9—Virtual and physical addresses

The page table for the system region is defined by the *system base register* which contains the physical address of the start of the system region page table and the *system length register* which contains the length of the table. Thus the system page table is contiguous in physical memory.

The per process space page tables are defined similarly by the *program* and *control region base registers* and *length registers*. However, the base registers do not contain physical addresses: rather, they contain system region virtual addresses. Thus the per process page tables are contiguous in the system region virtual address space and are not necessarily contiguous in physical memory. This placement of the per process page tables permits them to be paged and avoids what would otherwise be a serious physical memory allocation problem.

Access control

At a given point in time a process executes in one of four *access modes*. The modes from most privileged to least are called *kernel, executive, supervisor* and *user*. The use of these modes by VAX/VMS is as follows:

1. Kernel—Interrupt and exception handling, scheduling, paging, physical I/O, etc.
2. Executive—Logical I/O as provided by RMS.
3. Supervisor—The command interpreter.
4. User—User procedures and data.

The accessibility of each page (read, write, or no access) from each access mode is specified in the PTE for that page. Any attempt to improperly access a page is suppressed and control is transferred to an operating system procedure. The accessibility is assumed hierarchically ordered: if a page is writable from any given mode, it is also readable; and if a page is accessible from a less privileged mode, it is accessible from a more privileged mode. Thus, for example, a page can be readable and writable from kernel mode, only readable from executive mode, and inaccessible from supervisor and user modes.

A procedure executing in a less privileged mode often needs to call a procedure which executes in a more privileged mode: e.g., a user program needs an operating system service performed. The access mode is changed to a more privileged mode by executing a change mode instruction which transfers control to a routine executing at the new access mode. A return is made to original access mode by executing a return from exception or interrupt instruction (REI).

The *current access mode* is stored in the *processor status longword* (PSL) whose low order 16 bits comprise the PSW. Also stored in the PSL is the *previous access mode*; i.e., the access mode from which the current access mode was called. The previous mode information is used by the special probe instructions which validate arguments passed in cross access mode calls.

Procedures running at each of the access modes require a

separate stacks with appropriate accessibility. To facilitate this, each process has four copies of SP which are selected according to the current access mode field in the PSL. A procedure always accesses the correct stack by using R14.

In an earlier section, it was stated that the VAX-11 standard CALL instruction is used for all calls including those for operating system services. Indeed procedures do call the operating system using the CALL instruction. The target of the CALL instruction is the minimal procedure consisting of an entry mask, a change mode instruction, and a return instruction. Thus access mode changing is transparent to the calling procedure.

Interrupts and exceptions

Interrupts and exceptions are forced changes in control flow other than that explicitly indicated by the executing program. The distinction between them is that interrupts are normally unrelated to the currently executing program while exceptions are a direct consequence of program execution. Examples of interrupt conditions are status changes in I/O devices while examples of exception conditions are arithmetic overflow or a memory management access control violation.

VAX-11 provides a 31 priority level interrupt system. Sixteen levels (16-31) are provided for hardware while 15 levels (1-15) are provided for software. (Level 0 is used for normal program execution.) The current *interrupt priority level* (IPL) is stored in a field in the PSL. When an interrupt request is made at a level higher than IPL, the current PC and PSL are pushed on the stack and new PC obtained from a vector selected by the interrupt requester (a new PSL is generated by the CPU). Interrupts are serviced by routines executing with kernel mode access control. Since interrupts are appropriately serviced in a system wide rather than a specific process context, the stack used for interrupts is defined by another stack pointer called the *interrupt stack pointer*. (Just as for the multiple stack pointers used in process context, an interrupt routine accesses the interrupt stack using R14.) An interrupt service is terminated by execution of an REI instruction which loads PC and PSL from the top two longwords on the stack.

Exceptions are handled like interrupts except for the following: (1) since exceptions arise in a specific process context, the kernel mode stack for that process is used to store PC and PSL and (2) additional parameters (such as the virtual address causing a page fault) may be pushed on the stack.

Process context switching

From the standpoint of the VAX-11 architecture, the process state or context consists of:

1. The 15 general registers R0-R13 and R15.
2. Four copies of R14 (SP): one for each of kernel, executive, supervisor, and user access modes.
3. The PSL.
4. Two base and two limit registers for the program and control region page tables.

This context is gathered together in a data structure called a *process control block* (PCB) which normally resides in memory. While a process is executing, the process context can be considered to reside in processor registers. To switch from one process to another it is required that the process context from the previously executing process be saved in its PCB in memory and the process context for the process about to be executed to be loaded from its PCB in memory. Two VAX-11 instructions support context switching. The save process context instruction saves the complete process context in memory while the load process context instruction loads the complete process context from memory.

I/O

Much like the PDP-11, VAX-11 has no specific I/O instructions. Rather, I/O devices and device controllers are implemented with a set of registers which have addresses in the physical memory address space. The CPU controls I/O devices by writing these registers; the devices return status by writing these registers and the CPU subsequently reading them. The normal memory management mechanism controls access to I/O device registers and a process having a particular device's registers mapped into its address space can control that device using the regular instruction set.

Compatibility mode

As mentioned in the VAX-11 overview, compatibility mode in the VAX-11 architecture provides the basic PDP-11 instruction set less privileged and floating point instructions. Compatibility mode is intended to support a user as opposed to an operating system environment. Normally a compatibility mode program is combined with a set of native mode procedures whose purpose is to map service requests from some particular PDP-11 operating system environment into VAX/VMS services.

In compatibility mode the 16-bit PDP-11 addresses are

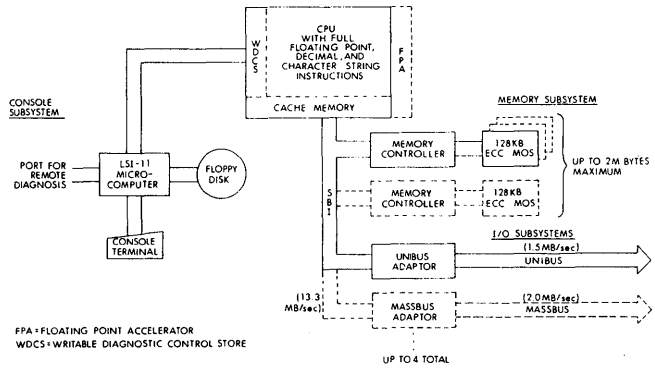


Figure 10—VAX-11/780 system

zero-extended to 32-bits where standard native mode mapping and access control apply. The eight 16-bit PDP-11 general registers overmap the native mode general registers R0-R6 and R15 and thus the PDP-11 processor state is contained wholly within the native mode processor state.

Compatibility mode is entered by setting the compatibility mode bit in the PSL. Compatibility mode is left by executing a PDP-11 trap instruction (such as used to make operating service requests), and on interrupts and exceptions.

VAX-11/780 IMPLEMENTATION

VAX-11/780

The VAX-11/780 computer system is the first implementation of the VAX-11 architecture. For instructions executed in compatibility mode, the VAX-11/780 has a performance comparable to the PDP-11/70. For instructions executed in native mode, the -11/780 has a performance in excess of the -11/70 and thus represents the new high end of the -11 (LSI-11, PDP-11, VAX-11) family.

A block diagram of the -11/780 system is given in Figure 10. The system consists of a central processing unit (CPU), the console subsystem, the memory subsystem, and the I/O subsystem. The CPU and the memory and I/O subsystems are joined by a high speed synchronous bus called the *Synchronous Backplane Interconnect* (SBI).

CPU

The CPU is a microprogrammed processor which implements the native and compatibility mode instruction sets, the memory management, and the interrupt and exception mechanisms. The CPU has 32-bit main data paths and is built almost entirely of conventional Shottky TTL components.

To reduce effective memory access time the CPU includes an 8K byte write through cache or buffer memory. The cache organization is 2-way associative with an 8-byte block size. To reduce delays due to writes, the CPU includes a write buffer. The CPU issues the write to the buffer and the actual memory write takes place in parallel with other CPU activity.

The CPU contains a 128 entry address translation buffer which is a cache of recent virtual to physical translations. The buffer is divided into two 64 entry sections: one for the per process regions and one for the system region. This division facilitates permitting the system region translations to remain unaffected by a process context switch.

A fourth buffer in the CPU is the 8-byte instruction buffer. It serves two purposes. First, it decomposes the highly variable instruction format into its basic components and, second, it constantly fetches ahead to reduce delays in obtaining the instruction components.

The CPU includes two standard clocks. The programmable real-time clock is used by the operating system for local timing purposes. The time-of-year clock with its own battery backup is the long term time reference for the operating system. It is automatically read on system startup to eliminate the need for manual entry of data and time.

The CPU includes 12K bytes of writable diagnostic control store (WDCS) which is used for diagnostic purposes, implementation of certain instructions, and for future microcode changes. As an option for very sophisticated users, another 12K bytes of writable control store is available.

A second option is the floating point accelerator (FPA). Although the basic CPU implements the full floating point instruction set, the FPA provides high speed floating point hardware. It is logically invisible to programs and only affects their running time.

Console subsystem

The console subsystem is centered around an LSI-11 computer with 16K bytes of RAM and 8K bytes of ROM (used to store the LSI-11 bootstrap, LSI-11 diagnostics, and console routines). Also included are a floppy disk, an interface to the console terminal, and a port for remote diagnostic purposes.

The floppy disk in the console subsystem serves multiple purposes. It stores the main system bootstrap and diagnostics and serves as a medium for distribution of software updates.

SBI

The SBI is the primary control and data transfer path in the -11/780 system. Because the cache and write buffer largely decouple the CPU performance from the memory access time, the SBI design was optimized for bandwidth and reliability rather than the lowest possible access time.

The SBI is a synchronous bus with a cycle time of 200 nsec. The data path width of the SBI is 32 bits. During each 200 nsec cycle either 32 bits of data or a 30-bit physical address can be transferred. Since each 32-bit read or write requires transmission of both address and data, two SBI cycles are used for a complete transaction. The SBI protocol permits 64-bit reads or writes using one address cycle and two data transfer cycles: the CPU and I/O subsystem use this mode whenever possible. For read transactions the bus is reacquired by the memory in order to send the data: thus the bus is not held during the memory access time.

Arbitration of the SBI is distributed: each interface to the SBI has a specific priority and its own bus request line. When an interface wishes to use the bus, it asserts its bus request line. If at the end of a 200 nsec cycle there are no interfaces of higher priority requesting the bus, the interface takes control of the bus.

Extensive checking is done on the SBI. Each transfer is parity checked and confirmed by the receiver. The arbitration process and general observance of the SBI protocol are checked by each SBI interface during each SBI cycle. The processor maintains a running 16-cycle history of the SBI:

any SBI error condition causes this history to be locked and preserved for diagnostic purposes.

Memory subsystem

The memory subsystem consists of one or two memory controllers with up to 1M bytes of memory on each. The memory is organized in 64-bit quadwords with an 8-bit ECC which provides single bit error correction and double bit error detection. The memory is built of 4K MOS RAM components.

The memory controllers have buffers which hold up to four memory requests. These buffers substantially increase the utilization of the SBI and memory by permitting the pipelining of multiple memory requests. If desired, quadword physical addresses can be interleaved across the memory controllers.

As an option, battery backup is available which preserves the contents of memory across short term power failures.

I/O subsystem

The I/O subsystem consists of buffered interfaces or adapters between the SBI and the two types of peripheral busses used on PDP-11 systems: the Unibus and the Massbus. One Unibus adapter and up to four Massbus adapters can be configured on a VAX-11/780 system.

The Unibus is a medium speed multiplexor bus which is used as a primary memory as well as peripheral bus in many PDP-11 systems. It has an 18-bit physical address space and supports byte and word transfers. In addition to implementing the Unibus protocol and transmitting interrupts to the CPU, the Unibus adapter provides two other functions. The first is mapping 18-bit Unibus addresses to 30-bit SBI physical addresses. This is accomplished in a manner substantially identical to the virtual to physical mapping implemented by the CPU. The Unibus address space is divided into 512 512-byte pages. Each Unibus page has a page table entry (residing in the Unibus adapter) which maps addresses in that page to physical memory addresses. In addition to providing address translation, the mapping permits contiguous transfers on the Unibus which cross page boundaries to be mapped to discontiguous physical memory page frames.

The second function performed by the Unibus adapter is assembling 16-bit Unibus transfers (both reads and writes) into 64-bit SBI transfers. This operation (which is applicable only to block transfers such as from disks) appreciably reduces SBI traffic due to Unibus operations. There are 15 8-byte buffers in the Unibus adapter permitting 15 simultaneous buffered transactions. Additionally there is a un-buffered path through the Unibus adapter permitting an arbitrary number of simultaneous un-buffered transfers.

The Massbus is a high speed block transfer bus used primarily for disks and tapes. The Massbus adapter provides much the same functionality as the Unibus adapter. The physical addresses into which transfers are made are defined by a page table: again this permits contiguous device transfers into discontiguous physical memory.

Buffering is provided in the Massbuss adapter which minimizes the probability of device overruns and assembles data into 64-bit units for transfer over the SBI.

ACKNOWLEDGMENTS

Although the final architecture is the result of several design iterations involving many hardware and software engineers, the author would like to acknowledge the other members of the initial architectural group: Gordon Bell, Peter Conklin, Dave Cutler, Bill Demmer, Tom Hastings, Richy Lary, Dave Rodgers and Steve Rothman. Mary Jane Forbes and Louise Principe deserve special thanks for typing this manuscript.

REFERENCES

1. McLean, J., "Univac Disbanding Future Systems Plan," *Electronic News*, December 12, 1977.
2. Bell, G., *et al.*, "A New Architecture for Minicomputers—the DEC PDP-11," *AFIPS Conference Proceedings*, Vol. 36, 1970.
3. Bell, G. and W. D. Strecker, "Computer Structures: What Have We Learned from the PDP-11," *Conference Proceedings: 3rd Annual Symposium on Computer Architecture*, 1976.
4. Myers, G. J., "The Case Against Stack-Oriented Instruction Sets," *Sigarch News*, August 1977.
5. Flynn, M. J., "The Interpretive Interface: Resources and Program Representation in Computer Organization," *High Speed Computer and Algorithm Organization* (Kuck, Lawrie, and Sameh, editors), Academic Press, New York, 1977.
6. Organick, E. I., *"The Multics Systems: An Examination of its Structure,"* MIT Press, Cambridge, 1972.
7. Schroeder, M. D. and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," *Proceedings Third Symposium on Operating Systems Principles*, 1971.
8. Needham, R. M., "Protection Systems and Protection Implementations," *AFIPS Conference Proceedings*, Vol. 41, 1972.
9. Needham, R. M. and R. D. H. Walker, "The Cambridge CAP Computer and Its Protection System," *Proceedings Sixth Symposium on Operating Systems Principles*, 1977.

APPENDIX—VAX-11 INSTRUCTION SET

Integer and Floating Point Logical Instructions

MOV-	Move(B,W,L,F,D,Q)*
MNEG-	Move Negated(B,W,L,F,D)
MCOM-	Move Complemented(B,W,L)
MOVZ-	Move Zero-Extended(BW,BL,WL)
CLR-	Clear(B,W,L=F,Q=D)
CVT--	Convert(B,W,L,F,D)(B,W,L,F,D)
CVTR-L	Convert Rounded(F,D) to Longword
CMP-	Compare(B,W,L,F,D)
TST-	Test(B,W,L,F,D)
BIS-2	Bit Set(B,W,L)2-Operand
BIS-3	Bit Set(B,W,L)3-Operand
BIC-2	Bit Clear(B,W,L)2-Operand

* B = byte, W = word, L = longword, F = floating, D = double floating, Q = quadword, S = set, C = clear.

BIC-3	Bit Clear(B,W,L)3-Operand
BIT-	Bit Test(B,W,L)
XOR-2	Exclusive OR(B,W,L)2-Operand
XOR-3	Exclusive OR(B,W,L)3-Operand
ROTL	Rotate Longword
PUSHL	Push Longword

Integer and Floating Point Arithmetic Instructions

INC-	Increment(B,W,L)
DEC-	Decrement(B,W,L)
ASH-	Arithmetic Shift(L,Q)
ADD-2	Add(B,W,L,F,D)2-Operand
ADD-3	Add(B,W,L,F,D)3-Operand
ADWC	Add with Carry
ADAWI	Add Aligned Word Interlocked
SUB-2	Subtract(B,W,L,F,D)2-Operand
SUB-3	Subtract(B,W,L,F,D)3-Operand
SBWC	Subtract with Carry
MUL-2	Multiply(B,W,L,F,D)2-Operand
MUL-3	Multiply(B,W,L,F,D)3-Operand
EMUL	Extended Multiply
DIV-2	Divide(B,W,L,F,D)2-Operand
DIV-3	Divide(B,W,L,F,D)3-Operand
EDIV	Extended Divide
EMOD-	Extended Modulus(F,D)
POLY-	Polynomial Evaluation (F,D)

Index Instruction

INDEX	Compute Index

Packed Decimal Instructions

MOVP	Move Packed
CMPP3	Compare Packed 3-Operand
CMPP4	Compare Packed 4-Operand
ASHP	Arithmetic Shift Round and Packed
ADDP4	Add Packed 4-Operand
ADDP6	Add Packed 6-Operand
SUBP4	Subtract Packed 4-Operand
SUBP6	Subtract Packed 6-Operand
MULP	Multiply Packed
DIVP	Divide Packed
CVTLP	Convert Long to Packed
CVTPL	Convert Packed to Long
CVTPT	Convert Packed to Trailing
CVTTP	Convert Trailing to Packed
CVTPS	Convert Packed to Separate
CVTSP	Convert Separate to Packed
EDITPC	Edit Packed to Character String

Character String Instructions

MOVC3	Move Character 3-Operand
MOVC5	Move Character 5-Operand
MOVTC	Move Translated Characters
MOVTUC	Move Translated Until Character
CMPC3	Compare Characters 3-Operand
CMPC5	Compare Characters 5-Operand
LOCC	Locate Character
SKPC	Skip Character
SCANC	Scan Characters
SPANC	Span Characters
MATCHC	Match Characters

Variable-Length Bit Field Instructions

EXTV	Extract Field
EXTZV	Extract Zero-Extended Field
INSV	Insert Field
CMPV	Compare Field
CMPZV	Compare Zero-Extended Field
FFS	Find First Set
FFC	Find First Clear

Branch on Bit Instructions

BLB-	Branch on Low B(S,Cl)
BB-	Branch on Bit(S,Cl)
BBS-	Branch on Bit Set and(S,Cl)Bit
BBC	Branch on Bit Clear and(Set,lear)Bit
BBSSI	Branch on Bit Set and Set Bit Interlocked
BBCCI	Branch on Bit Clear and Clear Bit Interlocked

Queue Instructions

INSQUE	Insert Entry in Queue
REMQUE	Remove Entry from Queue

Address Manipulation Instructions

MOVA-	Move Address(B,W,L=F,Q=D)
PUSHA-	Push Address(B,W,L=F,Q=D)on Stack

Processor State Instructions

PUSHR	Push Registers on Stack
POPR	Pop Registers from Stack
MOVPSL	Move from Processor Status Longword
BISPSW	Bit Set Processor Status Word
BICPSW	Bit Clear Processor Status Word

Unconditional Branch and Jump Instructions

BR-	Branch with(B,W)Displacement
JMP	Jump

Branch on Condition Code

BLSS	Less Than
BLSSU	Less Than Unsigned
(BCS)	(Carry Set)
BLEQ	Less Than or Equal
BLEQU	Less Than or Equal Unsigned
BEQL	Equal
(BEQLU)	(Equal Unsigned)
BNEQ	Not Equal
(BNEQU)	(Not Equal Unsigned)
BGTR	Greater Than
BGTRU	Greater Than Unsigned
BGEQ	Greater Than or Equal
BGEQU	Greater Than or Equal Unsigned
(BCC)	(Carry Clear)
BVS	Overflow Set
BVC	Overflow Clear

Loop and Case Branch

ACB-	Add, Compare and Branch(B,W,L,F,D)
AOBLEQ	Add One and Branch Less Than or Equal
AOBLSS	Add One and Branch Less Than
SOBGEQ	Subtract One and Branch Greater Than or Equal
SOBGTR	Subtract One and Branch Greater Than
CASE-	Case on(B,W,L)

Subroutine Call and Return Instructions

BSB	Branch to Subroutine with(B,W)Displacement
JSB	Jump to Subroutine
RSB	Return from Subroutine

Procedure Call and Return Instructions

CALLG	Call Procedure with General Argument List
CALLS	Call Procedure with Stack Argument List
RET	Return from Procedure

Access Mode Instructions

CHM	Change Mode to (Kernel,Executive,Supervisor,User)
REI	Return from Exception or Interrupt
PROBER	Probe Read
PROBEW	Probe Write

Privileged Processor Register Control Instructions

SVPCTX	Save Process Context
LDPCTX	Load Process Context
MTPR	Move to Process Register
MFPR	Move from Processor Register

Special Function Instructions

CRC	Cyclic Redundancy Check
BPT	Breakpoint Fault
XFC	Extended Function Call
NOP	No Operation
HALT	Halt

Pepe architecture—Present and future*

by C. R. VICK
U.S. Army Ballistic Missile Defence Advanced Technology Center
and
JOHN A. CORNELL
System Development Corporation
Huntsville, Alabama

INTRODUCTION

PEPE (Parallel Element Processing Ensemble) was designed, developed, and produced for the U.S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) for research on Ballistic Missile Defense (BMD) systems. With its host computer, the CDC 7600, it is potentially the world's most powerful computer system. PEPE employs parallel processing and associative data-access techniques, and while it can operate in a stand alone mode, it is designed primarily to augment more conventional computers in BMD service. Potential applications besides BMD for which the machine is adapted include weather forecasting, air traffic control, image data processing, signal processing, and other applications exhibiting inherent parallelism and requiring extensive computational power.

The PEPE system can be considered simply as a large master computer, called a host, controlling many smaller slave processors, called elements. In the present design, the host is a CDC 7600, and there are 288 elements. Each element comprises three processors sharing a common data memory. One of these processors, called a correlation unit, is used for inputting data and has an instruction repertoire especially suited for the rapid association of new data with data already on file. The second processor, called the arithmetic unit, has a repertoire similar to that featured in conventional high-power general-purpose machines; i.e., fixed and floating point arithmetic operations, load and store, and logical operations. The third processor, called the associative output unit, is used for ordering and outputting data and is especially designed to perform complex, multidimensional file searches rapidly and efficiently. Each of the three processors is driven by its own control unit, which simultaneously drives all of the corresponding processors in the ensemble of elements. The three control units are also capable of executing their own sequential programs. They are combined into a control console, which drives the ensemble of elements in parallel and interfaces the ensemble with the host. The complete PEPE/Host system, then, is a multiprocessor employing seven processor types in all (host, three sequential processors, and three parallel processors). All seven processor types are capable of simultaneous, overlapped operation.

Software for the PEPE includes the compilers and assemblers for the seven PEPE processors and a linkage editor for binding programs into executable load modules. The entire machine can be programmed in a single language called PFOR, which is a superset of FORTRAN. System software also includes an instruction-level simulator for PEPE, a general-purpose real-time operating system, and a general utilities package.

The PEPE program was a complete system effort, requiring problem analysis; generation of a hierarchy of system, functional, and implementation specifications; hardware design, development, production, and test; system support software design, development, production and test; tactical/problem software design, development, and test; and integration into a total real-time systems environment employing other computers, missiles, radars, and command, communications, and control networks. All of these activities were carried on concurrently. Moreover, much of the hardware architecture and programming techniques had never before been attempted on such a large scale for real-time service. It was obvious, therefore, that conventional procedures for executing a development program of this magnitude and in particular developing the PEPE architecture would not be adequate. Such procedures, employing the traditional sequence of problem analysis, design, hardware development, hardware fabrication and test, program design, development, production and test, and finally system test and validation, often result in major system errors being discovered in the last phase of the sequence when it is too late to do much about them. Because of the complexity of the PEPE development program, major system design errors would almost certainly occur (under the conventional pro-

* This work was supported by the U.S. Army Advanced Ballistic Missile Defense Agency, Huntsville, Alabama, under Contract DAHC60-73-C-0060. Portions of this paper were published in the 1976 Infotech Information Report on Multiprocessors.

cedures) and would propagate through the various stages of the program to be discovered in the final stage, system validation. To avoid the high risk inherent in the conventional procedures, PEPE system designers decided to employ a top-down design approach in which system validation started early and continued throughout the entire program. Thus, the program would be executed in a logical progression of *validated* baseline steps. The validation would be accomplished through a combination of functional and analytic simulations. Early in the project, the architecture and the operating system were validated via functional simulations which modeled the operation of the PEPE in a BMD environment employing a variety of attack scenarios. Further functional simulations of more complex environments employing PEPE in the National BMD Site Defense System were completed during the first half of 1975. A 36-element version of the PEPE hardware was delivered to the BMDATC Advanced Research Center in Huntsville in April 1976. Analytic simulations, employing the instruction-level hardware simulator and the PEPE hardware itself, were conducted during the last three years.

BMD DATA PROCESSING PROBLEM

Figure 1 portrays a simplified subset of the BMD data processing problem; only that part of the total problem associated with detecting, tracking, and classifying targets in the field of view of the radar is considered in this paper. The radar generates a pencil beam capable of being pointed, within a few microseconds, in any direction within the surveillance volume. Beams are either transmit or receive beams; each transmit beam can generate any one of a number of different pulse configurations (carrier frequency, number of pulses, pulse coding, frequency modulation, pulse length, etc., can be varied) and each receive beam can likewise assume any one of a number of configurations (carrier frequency, range extent, matched filter, etc., can be varied). Typically, the radar generates several thousand transmit beams per second and a proportionate number of receive beams. Generally, several hundred transmit and receive beam pairs are reserved for searching the upper part of the surveillance volume; these are generated in a raster scan pattern designed so that no object can penetrate the volume without being detected by at least one search beam. Interleaved at random in both time and space among the search beams, as required by the target environment and battle situation, are a variety of special beams for verifying search detections, precision tracking of previously detected and acquired targets, transmitting guidance commands to interceptors, target discrimination and classification, and several other functions. It is the job of the computer to initiate a file on each detected object, associate each radar return with its proper file, employ the information in each return to update the file, perform complex mathematical functions on the files, and generate requests for additional radar information. Each request becomes part of the file on a particular target. Periodically, the files are searched in some sort of priority fashion to generate orders for radar pulses. Details on each pulse are formulated individually by the computer based on the requests in the target files; those requests which are granted are transmitted to the radar in the form of orders. The objective is to select, for each pulse, that order which will be of most benefit to the defense system at that time. The radar transmits the ordered pulses, new returns are thereby obtained, and the radar computer loop is closed. The foregoing functions must all be performed within rather severe time deadlines. The BMD data processing problem, then, can be described as keeping a large, complex data base updated in real time. Moreover, the means for acquiring the information required for updating is under control of the computer.

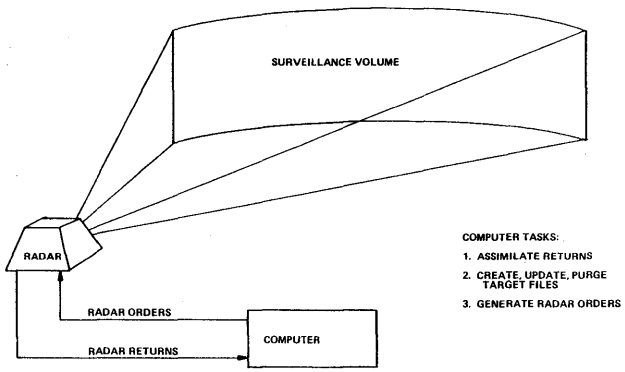


Figure 1—Partial BMD system

Considering the above rather simplified description of a representative part of the BMD data processing problem, one can deduce the types of processing required, and then synthesize a machine architecture to satisfy the requirements; this was done in the case of the PEPE design.

First, means must be provided to associate verified radar search returns with target data already on file in computer memory. Every such return must be compared with all target data on file. Each comparison, or association operation, requires a forward prediction of all filed target positions to the time of the return, calculation of three-dimensional gates (gate dimensions may be more or less than 3, depending on correlation criteria) about the positions, and a match of the radar return against all gate volumes.

A conventional sequential computer must execute the foregoing operations as an $m \times n$ number of operations, where m is the number of new returns in a given time interval and n is the number of targets on file. This means that the data processing resources required for correlation of new returns is an exponential function of the target traffic. Since correlation is one of the major consumers of sequential computer resources in BMD service, an architecture that performs correlation more efficiently is indicated. Associative processors perform the correlation function as an $m \times 1$ number of operations, independent of the number of targets on file. Therefore, a computer architecture for BMD service could advantageously possess associative data input characteristics.

Second, lengthy scientific computations must be made on a large number of separate, independent data sets, as in Kalman filtering of radar returns. Computations are identical

for all targets, so that considerable leverage could be obtained by assigning targets to separate, dedicated execution units operating on data in dedicated memories, all driven simultaneously from a single control unit executing a single program. Therefore, a computer architecture for BMD service could advantageously possess parallel processing characteristics.

Third, multidimensional, prioritized file searches must be made through target files for radar pulse requests, so that the radar pulses can be scheduled efficiently, in observance of a variety of constraints, and in accordance with the immediate needs of the instantaneous total battle environment. Such searches are made inefficiently in conventionally addressed memories, but very efficiently in associative memories. Therefore, a computer architecture for BMD service could advantageously employ associative data retrieval and output characteristics. Moreover, associative data retrieval would be of great assistance in one of the most critical and resource-consuming functions which must be performed in a total BMD system, a function largely ignored in this paper. The function is that of real-time control, or assigning resources and adjusting algorithms, procedures, function execution rates, and overall system behavior dynamically and optimally in accordance with instantaneous system demands, status, and defense strategy. This function, because it is sequential and decision-making in general, is best handled on a conventional sequential computer. However, the complex data interrogation functions which are necessary to efficient execution of the real-time-control task are greatly simplified when the sequential computer has associative access to the dynamic system data base.

In addition to the foregoing three primary BMD data processing requirements of correlation, scientific computation, and multidimensional file search (which are incidentally found to a similar degree in other problems), there are additional requirements common to military computers in general and to BMD computers in particular. First, the computer must be extremely reliable, and this reliability should preferably be an inherent characteristic of the architecture. High reliability is difficult to achieve in a computer suitable for BMD service, because of the very large amount of hardware such computers must contain. The latter is true because, in the final analysis, great computational power (estimates for BMD requirements run from tens to hundreds of millions of instructions per second) is achieved only via the employment of large amounts of CPU hardware. An architecture acceptable for BMD service must allow high reliability despite the large amount of hardware required. Parallel associative architectures provide opportunities for meeting this requirement, so long as the individual elements in the parallel array can be kept independent of one another. Then, individual elements can fail without affecting others, and without affecting the problem solution. Fortunately, many of the data sets dealt with in BMD computations are themselves independent, and lend themselves to assignment to independent processing elements. Since the data sets are independent, there is no need for interelement communication. This permits the arrangement of processing elements to be almost completely unstructured, so that no particular one or combination of elements is needed for successful problem completion.

A final architectural requirement, important for BMD applications, is that the data processing system be capable of easy expansion to meet increased requirements. A parallel, associative architecture, where the number and organizational structure of processing elements can be arbitrary, lends itself to this requirement too. Additional elements could be added to handle additional target traffic, and all of the programs could be written to be independent of the number and/or location of processing elements, so that programs need not be modified to accommodate hardware expansion.

Summarizing, the foregoing requirements led to consideration of a parallel, associative architecture. However, such computers which have been designed and/or built in the past fall generally into two categories, neither of which are really optimized for BMD service. First, past and contemporary associative processors are characterized by very primitive per-element computational capability (usually bit-serial); they achieve high processing speed by virtue of keeping a very large number of elements (say thousands) busy simultaneously. Computers for BMD service would not require the extremely large number of elements possible in such machines, but they do require fairly powerful elements because of the extensive per-target scientific calculations required. The second category is exemplified by the ILLIAC IV, which certainly has sufficient per-element computation capability, but an insufficient number of elements (tens), assuming that one wants to assign one target to an element (this is not really necessary, but not to do so leads to other complications and is beyond the scope of this paper). Moreover, ILLIAC IV contains features such as interelement communication which are not essential for BMD service, but which contribute to lower reliability because of the more or less rigid structure imposed upon the array by the intercommunication facilities. It would seem then, that the parallel associative architecture specified for BMD service should have capabilities somewhere between those offered by associative processors and ILLIAC IV. It should have a moderately large number of elements (hundreds), and each element should have a fairly powerful scientific computation capability. Moreover, each element should comprise three execution units, one each for input data correlation, parallel scientific computation, and associative file search for data output. To maximize throughput, all three execution units in the elements should be capable of simultaneous operation. The elements should have no positional significance; i.e., they should be organized into a completely unstructured array—an ensemble. Interelement communication and local indexing of data, while of considerable value in parallel computers designed to operate on interdependent data sets, can be eliminated because they are not needed and in fact are not even desirable. Finally and perhaps most important, the total data processing system should have high throughput in sophisticated branching and decision-making operations, but these operations should not reduce throughput on the less complex but resource-consuming parallel data processing functions. This means that the total data processing

system should have separate but closely cooperating facilities for both sequential and parallel processing tasks. The architecture which evolved from the foregoing considerations, PEPE, is described in the following section.

PEPE ARCHITECTURAL OVERVIEW

A system block diagram of the PEPE is shown in Figure 2. The Host, a CDC 7600, is connected to the three PEPE Control Units (collectively, the Control Console) via three standard CDC 7600 MUX (Input/Output Multiplexor) channels, one for each Control Unit. The CDC 7600 then sees the PEPE as three independent PPUs (Peripheral Processing Units). Each Control Unit is equipped with a modular Host Interface Unit; by replacing Interface Units, other interface connections with different Host machines are possible. The Host provides overall executive control, through its operating system, of the Host-PEPE configuration. It loads program and initial data into the PEPE memories, schedules and dispatches appropriate tasks to the PEPE, executes those sequential tasks which it does more efficiently than the PEPE, and executes utility functions such as compiling PEPE programs and reading and writing to peripheral devices.

The Control Console contains three independent Control Units (See Figure 3); they are similar but the Correlation Control Unit is optimized for inputting data to the elements, the Arithmetic Control Unit is optimized for scientific calculation in the elements, and the Associative Output Control Unit is optimized for outputting data from the elements. Each Control Unit is a multiprocessor which splits one instruction stream into two parts, one sequential which is executed within the Control Unit, and one parallel which is executed in the ensemble of elements.

Arithmetic control unit and parallel arithmetic units

The Arithmetic Control Unit is shown in block-diagram form in Figure 4. This unit contains a 32,768-word, 32 bit-

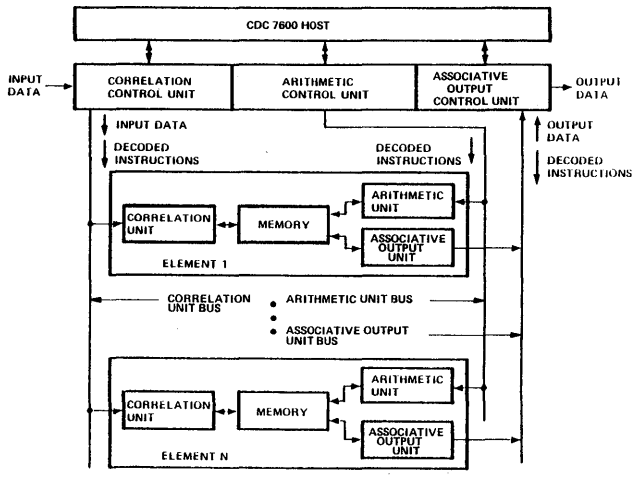


Figure 2—PEPE architecture

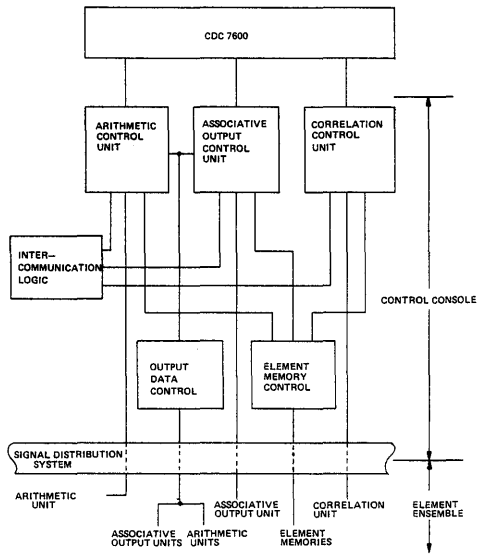


Figure 3—PEPE control console

per-word, random access semiconductor memory with a cycle time of 200 ns. A separate data memory contains 2048 random-access 32-bit words, implemented in ECL with a cycle time of 100 ns. The Sequential Control Logic (SCL) is a fairly conventional processor containing the usual A (Accumulator), B (operand), and Q (quotient/product) registers, plus 15 index registers. It executes integer (24-bit) arithmetic, and logical and branch instructions fetched from program memory on data obtained from data memory. Program memory contains a mixture of sequential and parallel instructions. The SCL executes the sequential instructions itself, but passes the parallel instructions, via the Parallel Instruction Queue (a conventional instruction stack) to the Parallel Instruction Control Unit (PICU). This unit decodes

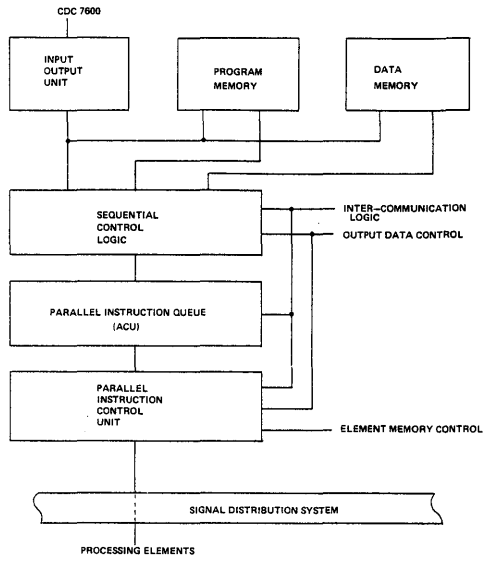


Figure 4—PEPE arithmetic control unit

the parallel instructions and broadcasts the resulting microoperation signals to all of the Arithmetic Units in the ensemble. The Arithmetic Units, provided they are active, all execute the same instructions simultaneously on data obtained from their own local data memories. The parallel instruction set is quite rich, containing the conventional arithmetic operations (plus square root) in both integer and floating point, load, store, and logical operations. In addition, the instruction set contains associative instructions which activate and deactivate elements based on element accumulator content and/or element tag register content. Only active elements execute instructions. Each Arithmetic Unit can execute about one million instructions per second, based on averages over instruction mixes encountered in typical scientific calculations.

Correlation control unit and parallel correlation units

The Correlation Control Unit (CCU) is similar to the Arithmetic Control Unit, but has a smaller, faster program memory and no Parallel Instruction Queue. Moreover, its PICU does not include a floating-point arithmetic capability, nor can it execute sequential floating-point operations. The CCU broadcasts a sequential stream of input data from the Host, or some other source, to all elements, and enters words from the input stream into properly activated elements. The Correlation Control Unit parallel instruction set is rich in associative match and compare operations, so that incoming data can be rapidly and efficiently correlated with data already in element memory, and thus be placed in the proper elements. Instructions are fetched from the CCU program memory and decoded, then the resulting microoperation signals are broadcast to the Correlation Units of all elements. The Correlation Units can be activated and deactivated independently of the element Arithmetic Units; only active Correlation Units execute instructions.

Each Correlation Unit contains a B register and 16 Correlation Registers which support execution of integer and logical operations, plus associative match and comparison operations among entering data and data retrieved from element memory. The Correlation Unit shares element memory with the Arithmetic Unit and the Associative Output Unit. Since the Correlation Unit does not perform floating point operations, its average instruction execution rate is about five million instructions per second, much faster than that of the Arithmetic Unit.

Associative output control unit and parallel associative output units

The Associative Output Control Unit (AOCU) is similar to the Arithmetic Control Unit, but, like the CCU, has a smaller, faster program memory, no Parallel Instruction Queue, and no capability for sequential or parallel floating-point arithmetic operations. Its function is to extract data sequentially from properly activated element memories, based on associative search instruction sequences which it fetches from its own program memory, decodes, and broadcasts (microoperation signals) to all element Associative Output Units (AOU). The AOUs can be activated independently of the element Arithmetic Units; only active AOUs execute instructions.

Each AOU contains conventional A and B Registers which support operation of integer and logical operations plus the aforementioned associative search operations. Parallel searches can be based on multidimensional search criteria, and data can be ordered in several different ways as they are obtained from the ensemble of elements. As in any associative memory, data output is sequential (in PEPE, sequential by 32-bit word). Data are buffered into AOCU Data Memory before transfer to the Host or some other device. Like the CUs, the AOUs execute instructions at an average rate of five million instructions per second, and they share element memory.

Internal interfaces

As shown in Figure 3, three major internal interfaces exist within the Control Console.

First, the Intercommunication Logic provides means for transferring data and control words among the control units. It contains a real-time clock and an interval timer accessible to all control units, and it provides means for the CCU and/or the AOCU to interrupt the ACU.

Second, the Element Memory Control receives requests from the three control units for element memory access. It performs any needed conflict resolution and transmits memory addresses to the elements.

Third, the Output Data Control is provided to resolve conflicts between AU and AOU requests to output data to the ACU and AOCU, respectively, over a common output data bus. The CUs have no capability for outputting data.

Design considerations

Several major design tradeoff decisions were made in assigning various functional capabilities to the different PEPE machine resources. These tradeoffs were made after extensive experiments involving simulated BMD exercises for a variety of attack scenarios and defense configurations. The tradeoffs were optimized for BMD service, and are not necessarily the ones that would have been made for PEPE application to other problems.

First, no provisions were made in the PEPE design for interelement communication. Data can be moved from element to element, but only through the Control Console, and the data transfer is sequential, one word at a time. This lack of a parallel interelement data transfer capability limits PEPE speed on problems involving computation performed on interrelated data sets (as encountered, for instance, in numerical solutions of partial differential equations), but for computations involving independent data sets, it is a real advantage. For the latter class of problem, each independent data set (for instance, a file on an aircraft, a missile, a

person) is assigned to an element, and each element performs identical operations simultaneously on the independent files. Since there is seldom, if ever, any interaction among the files, no interelement communication is required. The hardware simplification achieved by not providing interelement communication is obvious, but there are other more important advantages. The ensemble of elements can be completely unstructured (accordingly, the collection of elements is called an ensemble rather than an array); no element has any positional significance. Thus, the elements can be accessed, ordered, and grouped for data manipulation purposes purely on the basis of their contents, or associatively. An element can fail, with no impact on the calculations occurring in the other elements. Moreover, for many problems where data are periodically updated on the basis of continually arriving new data, only the historical state of an independent file is lost when an element fails. New data destined for that element are unable to correlate, so the data are automatically entered into an empty element to begin a new file. Thus, graceful degradation and automatic recovery are achieved naturally. Further, elements can be added as required with no effect on software or program execution. In fact, PEPE programs are oblivious of the number of elements, and programmers do not know or care either, as long as there are enough.

Another design decision involved floating-point versus fixed-point arithmetic. This decision had to be made for each of the six different processor types in PEPE (one sequential and one parallel processor for each control unit). The sequential processors in the control units are used mainly for comparison, branching, control of program flow, and supervisory operations, all of which can be executed with only integer, branching, load, store, and logical operations. Therefore, no floating-point hardware was provided in the control unit sequential processors. The Arithmetic Units in the elements have scientific calculations as their primary responsibility; therefore, they were provided with a full, powerful, floating-point arithmetic instruction repertoire in addition to all of the other conventional and associative grouping instructions. The CUs and AOUs are mainly required to input/output data and perform associative comparisons, matches, searches, and ordering functions, all of which can be handled with logical and integer arithmetic operations. However, they are occasionally called upon to execute short subroutines containing floating-point operations. Rather than provide expensive floating-point hardware in all of the CUs and AOUs to accommodate the rather infrequent requirements, it was decided to include instead provision for the CCU and AOCU to interrupt the ACU. Then, the ACU could perform the floating-point routines on demand for the CCU and the AOCU. A considerable amount of hardware was thereby saved, and simulations show that the performance degradation caused by lack of floating-point capability in the CUs and AOUs is insignificant.

A third design decision involved the Parallel Instruction Queue, which was provided only in the parallel instruction stream in the ACU. It was provided there because the ACU program memory is much larger and therefore slower than the other PEPE memories (200 ns vs 100 ns cycle time). Also, the average ACU instruction time is one microsecond, so that ACU parallel program execution time could be significantly reduced by inserting a fast stack (the PIQ has 16 ECL registers) between the program memory and the PICU. Since the CCU/CU and AOCU/AOU average instruction speed is 200 ns and the smaller CCU and AOCU program memories have cycle times of 200 ns, no such speedup could be realized in these processors with instruction stacks, and none were provided.

A fourth decision involved the output data bus from the parallel element to the Control Console. Although most element data are output from the AOUs to the AOCUs, occasionally the AUs contain data in registers destined for output to the ACU. To accomplish this transfer, the data could be stored back into element memory, retrieved by the AOU, output to the AOCU, and then transferred to the ACU. Alternatively, a separate AU/ACU data bus could be provided. Since the first alternative was too slow and the second too expensive, it was decided to allow the AUs and AOUs to share the same output data bus, and logic to implement this was provided. Since the AUs output data infrequently relative to the AOUs, output operations are not significantly degraded.

A final decision arose because the element CU, AU, and AOU share a common data memory, and since they can all operate simultaneously, conflicts inevitably occur and must be resolved. Since the element memory has a cycle time of 100 ns, and the average instruction time is one microsecond for the AU and 200 ns for both the CU and the AOU, it would seem that either the CU or the AOU should have first priority. This priority, however, is problem-dependent so simulations were held to establish priorities for memory conflict resolutions. The simulations demonstrated that the priorities should be CU first, AOU second, and AU last. With this priority assignment, the AU program execution time was extended only about five percent over what it would be if the AU had exclusive memory access.

Since memory access conflicts are rare (for typical BMD problems) and since the CU, AU, and AOU can operate simultaneously, average instruction rates for a single element can approach 11 million instructions per second. When all elements are operating, an ultimate rate of over a billion instructions per second can be achieved, but this rate is highly problem-dependent and would occur infrequently.

PEPE APPLICATION IN BMD

A simplified subset of the BMD data processing problem, as implemented on PEPE for its solution, will be described here to convey an idea of how PEPE is used in real-time control applications. Extensions to other similar data processing problems should be obvious, or at least not difficult. The problem comprises the maintenance of data files in real time on all targets (typically several hundred) within view of the phased-array radar. This requires matching each radar return against all target files to determine which file should accept the return data, and placing the return data in that file (typically ten to a hundred instructions per return per

file). If no match is possible, the return is assumed to be from a new target and is used to initialize a new file. Then, the new return is used to update the target file (typically several thousand instructions per update). Finally, based on the updated file, a request for a subsequent radar pulse to gather new data on the target is generated (typically tens to several hundred instructions per update). The request is then scheduled on the radar time line based on a multiplicity of complex radar/environmental/situation constraints (typically several hundred instructions per file per request). The real-time constraint is that, for each return (several hundred to several thousand per second), a request for a subsequent radar pulse must be generated and delivered to the radar within a specified time interval. The time interval, generally called port-to-port time, is usually different for different functions and can be extremely critical for some of them.

The problem is implemented on PEPE by assigning each individual target and its file to one element (other implementations are possible but this is most straightforward). For simplification, initialization of files and starting the problem on the PEPE will be ignored; instead the course of the problem will be picked up in the middle of a BMD engagement and carried on from there. Several hundred target files occupy the ensemble memory, one target file to an element. Periodically, the files are updated in parallel by the ACU/AUs using appropriate mathematical routines. Generally, time enablement of the routines is used; the frequency of enablement is chosen high enough so that port-to-port timing restrictions are not exceeded. For any one enablement, not all files are updated; only those elements which have received new radar return data since the last update are activated and only they participate in the parallel updating.

While the file updating is proceeding in the ACU/AUs, radar return data are streaming sequentially into the CCU data memory, either through the host or directly from the radar. As each return enters (a block of about ten 32-bit words describing target position, time of return, etc.) the CCU interrupts the ACU. The ACU then executes a prediction routine which predicts forward, to the time of the return, the positions of all targets on file. Then, the ACU constructs multidimensional windows around the positions. These operations are performed simultaneously for all targets on file. The ACU is then released, and the CCU orders the transfer of all of the window parameters into the CU correlation registers. The new return data are broadcast to all CUs, and the target position is compared simultaneously to all windows. Where a comparison is successful, the return data are placed in the element where success was achieved, and are used to update the target file during the next AU update interval. When no successful comparisons are made, the target is assumed to be detected by the radar for the first time, and the return data are placed in an empty element to start a new target file. This sequence of events is continuous, and it occurs simultaneously and asynchronously with the updating operations taking place in the AUs. The sophisticated reader will note that the foregoing procedure is a gross simplification of the target correlation problem, no account being given for practical considerations such as crossing targets, ghosts, and redundant targets. However, such problems are common to all computers and depend upon algorithms, not architecture, for their solution. As expected, the associative properties of PEPE make it an especially efficient implementer of algorithms designed to solve practical target correlation problems.

As a final operation in each update occurring in the AUs, requests for subsequent radar transmit/receive pairs, one request per target, are generated. Request formats include time of pulse, type, beam direction cosines, range windows, special pulse characteristics, etc., depending upon target type, priority, and a variety of other considerations. Resolution of conflicting pulse requests, allocation of pulses to targets, and scheduling of requests in the form of pulses and receive windows on the radar time line, are tasks assigned to the AOCU/AOUs. Generally, the objective is to schedule each pulse so as to maximize the BMD system response at the current instant, subject to a large variety of constraints and demands. This requires that target files be searched for highest-priority requests, where priority is a function of many variables. These requests must then be matched against a set of dynamic constraints (generally stored in the Host) in order to build a block of radar orders. Obviously, the procedure calls for complex multidimensional file-searches and matching and comparison operations, and is extremely time and resource-consuming for conventional sequential computers with location-addressed memories. Since the AOCU/AOU conducts the file searches and makes the matches and comparisons associatively, the procedure is considerably simplified and speeded up.

PEPE CONSTRUCTION

PEPE construction is rather straightforward, and employs circuit devices, hardware, structure, and wiring which may be considered state-of-the-art for supercomputers. The Control Console is contained in one cabinet 84″ high, 82″ wide, and 30″ deep. Elements are contained in identical cabinets, 36 elements to a cabinet. Eight element cabinets therefore comprise a full 288-element ensemble.

A complete element is contained in six 16″×18″ printed circuit plug-in boards, two each for the AU and the AUO, and one each for the memory and the CU. All board wiring is contained in eight printed circuit layers, four for signals and four for voltages and ground. Signal layers are designed to maintain a constant 50-ohm impedance. The boards connect to cabinet back-plane wiring through four plug-in connectors per board, 100 pins per connector. All logic is implemented in MECL 10K Dual-in-Line (DIL) integrated circuit packages which plug into sockets on the boards, Each board has space for 300 DILs, but there are only about 1000 DILs per element.

The boards plug into the cabinets vertically in four rows, nine elements to a row. Each row has its own power supply which supplies 5.2 volts at 560 amperes and 2.0 volts at 540 amperes to all of the elements in its row. The right-hand side of the cabinet contains these four power supplies arranged vertically, and the left hand side contains signal distribution

logic and circuitry. The four backplanes (one for each row) into which the elements are plugged are fabricated from three heavy laminated copper plates, which provide dc power distribution, signal ground plane, and power-supply bypass capacitance. Backplane interelement signal wiring is implemented entirely in 50-ohm subminiature coaxial cable. All of the nine elements in a row are connected to one signal distribution bus comprised of about 300 separate balanced-pair high-speed signal lines (PEPE uses a 10-mHz clock). The bus consists of flat belted cables which maintain constant impedance throughout the signal distribution system. The cables are connected to paddle boards which plug into the rear of the back plane to make connection to the elements. The four busses in a cabinet are fanned out from signal distribution circuitry in the left end of the cabinet. The eight cabinets are all connected to the Control Console signal distribution system via balanced-pair, constant-impedance flat belted cables.

Cooling is provided by a water-cooled chill plate located under the bottom row in the cabinet, and forced-air dual blowers located in a plenum beneath the cabinet. Total power dissipation is about 20 kW per element cabinet.

The Control Console cabinet construction is similar to that of the element cabinet, except that the Control Console printed-circuit boards use only six printed circuit layers, plus some conventional wire-wrap signal interconnection.

PEPE SUPPORT SOFTWARE

A detailed discussion of PEPE software is beyond the scope of this paper, but a summary of the capabilities of the support software system developed for PEPE will be given here. The support software system comprises a real-time executive system, a PEPE operating system, an instruction-level hardware simulator, a procedure-oriented language and translation system, and a general utilities package. All of the foregoing operate under control of the CDC 7600 SCOPE 2.0 Operating System.

The real-time executive has two primary responsibilities. First, it schedules tasks for execution on the host and the various PEPE processors in accordance with requests generated by other tasks, data enablements, time enablements, and system status. Second, it dispatches tasks in accordance with priorities and satisfied precedence relationships. The real-time executive resides and operates in both the CDC 7600 Host and the PEPE ACU.

The PEPE Operating System, which includes the real-time executive and supplies support service for it, has components which reside and operate in the CDC 7600 Host and all three control units. It includes mainly interrupt handlers and input/output handlers. Data buffers for Host/Control Console input/output operations, as well as storage space for the process control tables which support the real-time executive are also provided in the CDC 7600 small-core memory and the data memories in the PEPE control units.

The instruction-level hardware simulator is a program which runs on the CDC 7600 and executes interpretively on real PEPE problem code to simulate the operation of the PEPE/Host configuration (except, of course, for run time). It was produced to support the development of PEPE system and problem software prior to hardware availability, but it has additional important uses. Since the initial PEPE hardware installation will comprise only 11 of the 288 elements, it will be used in combination with the hardware to run tests where 11 elements are insufficient. Also, since the entire PEPE Project is at present an experimental one, the simulator will be used to check proposed hardware modifications before they are implemented.

PEPE is programmed in PFOR, a procedure-oriented, higher order superset of FORTRAN. PFOR also includes PAL, the PEPE Assembly Language. In summary, PFOR adds to FORTRAN additional statements which permit the declaration of parallel variables (those variables stored in the element memories), and which implement and exploit the parallel processing and associative access/grouping properties of PEPE. The PFOR compiler is a set of compilers and assemblers, together with a linkage editor, which accepts a single PFOR source program and expands it into object programs and load modules for all of the PEPE processors. Current experience shows that PFOR is an easy language to use and PEPE is easy to program; FORTRAN programmers can learn to program PEPE in a week or so. PEPE programs are generally characterized by a structural simplicity which makes them easy to read, understand, debug, and modify. Of particular interest is the relative absence of complicated program loops and housekeeping operations on data locations.

The PEPE Utilities package is quite conventional; it comprises the usual loaders, debug aids, and data recording programs. However, the actual implementation of the package components is unique because of the PEPE parallel/associative architecture.

PEPE IN NON-BMD APPLICATIONS

BMD data processing can be characterized as parallel computation on independent data sets; this characterization can also be applied to air defense, air traffic control, and several other applications. Such applications, when implemented on parallel architectures, do not benefit from a capability for parallel interelement data transfer among the parallel elements; in fact, there are disadvantages, as discussed previously in this paper, in providing such a transfer capability. However, there is a large class of problems which require operations on interdependent data sets, such as weather forecasting and other fluid dynamics problems requiring the numerical integration of nonlinear partial differential equations. Such problems can be solved much faster when implemented on parallel architectures, but parallel interelement transfer capability seems, at first look, to be required for efficient execution. In this section of the paper, the operation of PEPE, which has no provision for parallel interelement data transfer, on such problems will be discussed.

In fluid dynamics problems, interest focuses on the time-

variant behavior of several dependent variables at a multiplicity of fixed points in 3-dimensional space. This behavior is described (for a continuum) by a set of partial differential equations, generally nonlinear. The only practical procedures for integrating these equations are numerical ones, in which the region of interest is divided into a 3-dimensional grid, and the partial differential equations are replaced by their linear-difference approximations at each point in the grid. The resulting system of linear equations is then solved by any one of a number of methods developed for that purpose. Generally, these methods involve iteration and most of them fall within a broad class of operations called relaxation techniques. Computer architects do not need to know details of the specific algorithms used. It is sufficient to know that the relaxation procedure is started in a computer by assigning to each interior grid point estimated values for the dependent variables. Further, variables at exterior grid points are assigned values fixed by boundary conditions or if not, estimated values. Then, for each grid point, the computer performs several hundred to several thousand mathematical operations (only the instruction mix and count are of interest to a computer architect) to "update" the values of the dependent variables at that point. These operations involve the variables at that point, and all the variables at the surrounding six points. After each grid point in the total volume of interest is thus treated (one relaxation), there are better estimates of the variables at all the grid points. The computer then repeats the whole procedure for as many relaxations as are required to achieve convergence. The foregoing is a simplified and incomplete description of how a computer proceeds to solve problems in fluid dynamics, but it is sufficient for this paper.

A sequential computer can operate on only one grid point at a time; therefore, run times can be extraordinarily long. Obviously, there is a considerable amount of parallelism in relaxation techniques of the type described above, so one should be able to do better with a parallel processor. The most direct approach would be to assign an element to each grid point, store the initial values of the variables at each grid point in its assigned element, and carry out all of the grid-point computations simultaneously. There are, however, at least two problems which make this approach impractical. First, there are requirements for fluid dynamics calculations in regions containing several hundred thousand grid points, and this is not a practical number of elements.

Second, each element needs variables from other elements to perform its calculations, and moving these variables into the proper elements prior to calculations for each relaxation is a formidable engineering problem if a large number of elements are involved.

The first problem can be solved by employing a mass storage device to store a complete grid image, and to have the parallel processor operate on one section of the grid image at a time. Thus, the variables at the grid points in a section of the grid image would be transferred to the parallel elements, calculations would be performed on all of the grid points in that section, and the updated variables would be transferred back to that section in the grid image to replace the old values. Moreover, variables representing more than one grid point could be stored in an element, the number depending on the size of element memory. A complete sweep through the entire grid image would be made per relaxation, with the size of each section depending on the number of parallel elements and the number of grid points assigned per element. Sweeping through the grid image on mass storage and in effect replacing the image with a better estimate of it represents a lot of data transfer between the mass store and the parallel processor, but it cannot be avoided as long as there are not enough elements to contain the entire grid image, which will almost always be the case for the sizes of problems faced by fluid dynamicists. This data-transfer operation, incidentally, is independent of the second problem and is encountered in all parallel processors whether or not they have efficient facilities for transferring data among elements. The mass store data transfer problem must be handled by suitable choice of mass store and its interface to the parallel processor.

The second problem can be handled in several different ways. First, provision can be made in the hardware for efficient interelement transfer, as was done in the ILLIAC IV. This complicates the hardware and forces a rigidly structured implementation of the problem on the processor array, which in turn leads to reliability problems and perhaps topographical mapping problems with grids of unconventional geometry.

Second, an external data manipulator, or permuter, can be attached to the parallel processor array to allow parallel transfer of data from any ordering of elements to any other ordering of elements. This is the approach taken by the STARAN flip network and Feng's "Versatile Data Manipulator." It requires a lot of extra hardware, but can handle any grid geometry.

Third, interelement data transfer can be dispensed with entirely, as in the PEPE, so long as all of the data needed for one relaxation calculation is entered into the proper elements at the time a grid section is transferred from mass memory to the parallel processor array. This requires more storage per element, since each element must store not only the variables it is responsible for updating, but also all those additional variables needed to do the updating. This could mean a factor of up to six times the element storage required by the other two schemes. However, the hardware is simple, complicated grid geometries are no problem, and the parallel array could employ an associative data transfer scheme between the mass storage and the array to counter the reliability problem of failed elements. This scheme would allow use of an essentially unmodified PEPE in fluid dynamic problems.

Assume a global circulation problem, in which we place on the globe a grid point at every five degrees of latitude and longitude, and at six altitude levels. Unwrapped from the globe, this is a rectangular $72 \times 36 \times 6$ grid. At each of these grid points, we are interested in the behavior of eight dependent variables. We will, therefore, be performing calculations at each grid point iteratively to continuously update the values of the variables. To perform one update of the eight variables at one grid point requires that we have available the previous eight values at that point and the

current 48 values from the six neighboring points (north, south, east, west, above, below), and that we must perform 2000 arithmetic operations for each grid point. One complete relaxation requires an update of all $36\times72\times6=15{,}552$ grid points. All of the foregoing numbers are realistic, being derived from the NCAR Global Circulation Model.

To solve the above problem on PEPE, we store the entire grid image on mass storage. We assume the mass storage is connected, via simple interface, to the CCU and the AOCU. Alternatively, it could be connected to the host via a standard interface. Now our general approach is to compute on a section of the grid at a time, to always store the required neighboring grid points in the same element as the point being updated so that interelement data transfer is never needed, and to simultaneously compute, input, and output on three different grid sections. Specifically, for each grid-section relaxation we will assign to each element responsibility for updating the six points in one column. That means we store in each element one column and the four neighboring columns, or $5\times6\times8=240$ words. Now we have all the data needed to update the six points in the center column. Since PEPE will be simultaneously computing on one grid section, inputting from mass store on the next, and outputting to mass store the updated values of the previous section, we need $3\times240=720$ words of storage per element.

To update the center column requires 2000 instructions per grid point, or $6\times2000=12000$ per column. Since the PEPE AU computes at a rate of one MIP, this takes 12 ms. Now, how many columns can be updated simultaneously, or stated another way, how many elements can be kept busy? The answer will give the performance that can be obtained from PEPE, or the number of MIPS that can be applied to the problem.

As soon as one set of columns is updated, computation should start on the next set of columns, to keep the AUs busy continuously. That means the loading of the next five columns in the next grid section, and the unloading of the previous five columns should be complete in 12 ms. Unloading (center column only) of 48 words/element at 2.4 μs/word=115.2 μs/element (2.4 μs/word is the AOU to AOCU transfer time; we assume that AOCU to mass memory time can keep up with this). Therefore $12000/115.2=107$ elements can be unloaded during the 12 ms compute time.

Loading is a little more complicated. Five columns per element must be loaded in 12 ms, but five elements can be loaded simultaneously, provided we can steer the columns to the right elements. If we can do this perfectly, then loading time will be the same as if we only had to load one column per element. Assume we cannot do it perfectly, so that we have to load the equivalent of two columns per element. Then, loading takes 96 words/element at 1.2 μs/word, so 107 elements can be loaded during the 12 ms compute time (1.2 μs/word is the CCU to CU transfer time).

All of the foregoing means we can keep 107 elements concontinuously busy, giving a computational rate of 107 MIPS on the global circulation problem. This is better than any other existing machine can achieve, especially when one considers that the MIPS required for I/O are overlapped with the compute MIPS and do not subtract from them. In other machines, of equivalent MIP rating, the I/O MIPS (and/or interelement transfer MIPS) must be subtracted, which effectively lowers computational rates.

There are several advantages to the foregoing implementation. The formulation is independent of grid size and geometry; these can be changed easily. The implementation is straightforward and should be easy to program. Data management should be simple, especially when compared to that required for sequential and vector machines. The use of associative input to the elements should permit element failures without program abort, so throughput per shift should be higher than can be obtained from less reliable machines of equivalent MIP rating. Finally, the more sophisticated and complex the updating algorithms are (which is the trend), the better the performance. For instance, 4000 instructions (rather than 2000) would result in keeping 214 elements continuously busy, or 214 MIPS.

ARCHITECTURAL ENHANCEMENT TO THE PEPE

The major architectural improvement that can be made in the present PEPE design is simplification of the signal distribution system. Such a modification would result in a capability for incorporating recent technological advances, primarily the use of available bit-slice microprocessor chips, into the element design. It would also result in an ability of several subsets of the PEPE ensemble to be executing programs at the same time, instead of only one as in the present design. The signal distribution system of PEPE is also the limiting physical factor involved in increasing the number of elements in a parallel processor. Studies show that the PEPE signal distribution system (drivers, receivers, logic transmission lines, connectors) can be simplified about two orders of magnitude. The price paid is some additional complication in the elements. To understand this, consider the original concept of PEPE. A single control unit, containing instruction memory, and instruction fetching, interpretation, and decoding circuitry drives a large number of execution units, each with its own memory. This concept of multiple execution units and only one control unit saves a great deal of hardware in a parallel processor, because the control unit in a computer generally contains a significant amount of complicated hardware. However, the interface between the control unit and the execution unit is the most complicated interface in a computer, and it is this interface which, under the original PEPE concept, must be carried by the signal distribution system to all elements. At the time of the original PEPE concept, engineering tradeoffs favored the single control unit and the complicated distribution system.

Tradeoffs would produce different results now. With thousands of elements, one probably cannot tolerate a complicated signal distribution system. Moreover, control unit circuitry is much cheaper; in fact, microprocessors contain both instruction decoder and execution unit on a single chip. The interface between the instruction decoder and the execution unit is not even available outside the chip.

So, one can put both the instruction decoder and the execution unit in the element much easier and cheaper than

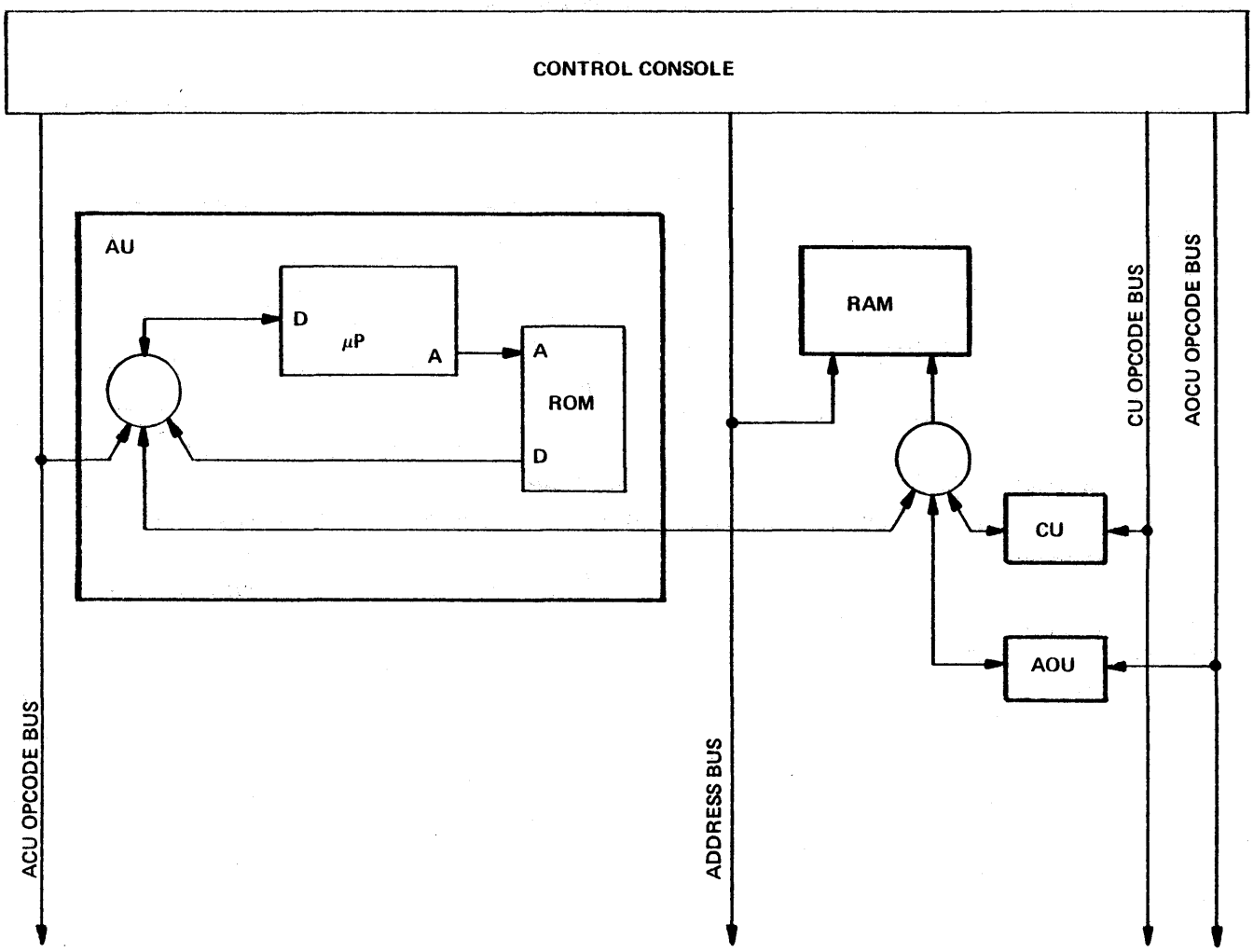


Figure 5—New PEPE scheme

was possible a few years ago, by making extensive use of LSI in the elements (possibly even off-the-shelf microprocessor chips). Then, the PEPE control unit would contain only a program memory and instruction fetching circuitry. This control unit broadcasts only the opcode to the elements, rather than decoded opcode signals. Fewer than ten signal paths are required to broadcast opcodes, rather than a hundred or so, as is the case when the decoded opcode must be broadcast. Moreover, the signal levels on the signal distribution system must change only once every instruction, rather than once every clock pulse. Thus, for the same instruction rate, the switching signals on the signal distribution system can be about ten times slower. These two factors, fewer signal paths and slower switching speeds, will permit great simplification in the signal distribution system.

Figure 5 shows the scheme. Only the details of the AU are shown, although the CU and AOU can be handled identically. The AU contains a microprocessor chip, which contains both instruction decoder and execution unit. In operation, the simplified ACU in the control console fetches an instruction, and places the address on the element address bus and the opcode on the opcode bus. The opcode first enters the data pins on the microprocessor chip, then the data from element memory is entered into the same pins. The instruction is decoded and executed on the chip and the result is returned to element memory. This sequence is conventional for almost all microprocessors. A further refinement can be made by storing special subroutines, such as trigonometric or exponentials, in ROM in the AU. Thus, the control console can include these functions within its opcode set and slow down the signal distribution system switching speed even more, while actually increasing the PEPE instruction rate.

Other useful operations can be done with a PEPE configured as described above. Since the ACU executes on the average about ten microinstructions for every opcode, more than one subset of AUs (same statement goes for CUs and AOUs) can be active at the same time, which is not possible with the present PEPE design. All that is needed is for the opcode bus to carry an activity number with it; only that set

of elements holding that activity number would execute that opcode. Then, while the selected subset of elements is executing the instruction, another instruction could be broadcast on the same bus to a different subset of elements.

This type of operation (it could be viewed as the logical equivalent of two or more ensembles) would keep more elements busy at the same time. Even further, the ensemble elements could store complete subroutines in their program memories, so that the control units need only broadcast subroutine calls rather than instructions. This would decrease signal-distribution system traffic even more, and would allow further exploitation of the ability of the subsets of the ensemble to execute several operations simultaneously.

REFERENCES

Lee, C. Y., and M. C. Paull, "A Content Addressable Distributed Logic Memory with Applications to Information Retrieval," *Proc. IEEE,* Vol. 51, June 1963, pp. 924-932.

Crane, B. A., and J. A. Githens, "Bulk Processing in Distributed Logic Memory," *IEEE Trans. on Electronic Computers,* Vol. 14, April 1965, pp. 186-196.

Huttenhoff, J. H. and R. R. Shively, "Arithmetic Unit of a Computing Element in a Global, Highly-Parallel Computer," *IEEE Trans. on Computers,* Vol. C-18, August 1969, pp. 695-698.

Githens, J. A., "An Associative Highly Parallel Computer for Radar Data Processing," Chapter 3 of *Parallel Processor Systems, Technologies, and Applications,* L. C. Hobbs, D. J. Theis, Joel Trimble, Harold Titus, and Ivar Highberg, Spartan Books, 1970, pp. 71-86.

Crane, B. A., M. J. Gilmartin, P. T. Rux, and R. R. Shively "The PEPE Computer," *IEEE Compcon 72 Digest.*

Wilson, D. E., "The PEPE Support Software System," *IEEE Compcon 72 Digest,* Sept. 1972, pp. 61-64.

Cornell, J. A., "PEPE Applications and Support Software," *IEEE Wescon 72 Digest,* Sept. 1972.

Bergland, G. D. and C. F. Hunnicutt, "Application of a Highly-Parallel Processor to Radar Data Processing," *IEEE Transactions on Aerospace and Electronic Systems,* March, 1972.

Cornell, J. A., "Parallel Processing of Ballistic Missile Defense Radar Data with PEPE," COMPCON 72 Computer Conference, 1972.

Johnson, M. D., "The Architecture and Implementation of a Parallel Element Processing Ensemble," Western Electronics Show and Convention, 1972.

Evensen, A.J., and J. L. Troy, "Introduction to the Architecture of a 288-Element PEPE," 1973 Sagamore Computer Conference on Parallel Processing.

Dingeldine, J. R., H. G. Martin, and W. M. Patterson, "Support & Operating System Software for PEPE," 1973 Sagamore Computer Conference on Parallel Processing.

Troy, J. L., "Computer Simulation of PEPE and its Host at the Instruction Level," 1973 Sagamore Computer Conference on Parallel Processing.

Barrett, A. L., "Process-Construction for a Parallel-Sequential Computer Architecture," 1973 Sagamore Computer Conference on Parallel Processing.

Berg, R. O., H. G. Schmitz, and S. J. Nuspl, "PEPE-An Overview of Architecture, Operation, and Implementation," *Proceedings, National Electronics Conference,* Chicago, Ill., October 1972.

DiVecchio, M., "The Design and Implementation of A High/Low Magnitude Search Instruction on PEPE," 1975 Sagamore Computer Conference on Parallel Processing.

Merwin, R. E. and C. R. Vick, "An Architectural Description of A Parallel Element Processing Ensemble," International Symposium on Computer Architecture, Grenoble, France, 1973.

Marshall, D. D., "A Parallel Processor Approach to Solving Decision Trees," 1977 International Conference on Parallel Processing.

Welch, H. O., "Numerical Weather Prediction in the PEPE Parallel Processor," 1977 International Conference on Parallel Processing.

Blakely, C. E., "PEPE Application to BMD Systems," 1977 International Conference on Parallel Processing.

Evensen, A. J., "PEPE Hardware and System Overview," 1977 International Conference on Parallel Processing.

PEPE—A user's viewpoint; a powerful real time adjunct

by M. P. MARIANI and E. J. HENRY
TRW—Defense and Space Systems Group
Redondo Beach, California

INTRODUCTION

The Parallel Element Processing Ensemble (PEPE) offers a cost-effective means of obtaining significant improvements in data driven real time systems. The inherent parallelism and programmability of the PEPE architecture make it an extremely effective growth option for data driven systems developed around a large-scale centralized Host. With the PEPE architecture, incremental system growth with minimum software impact can become a reality.

The purpose of this paper was to investigate the implementation of PEPE in real time data driven systems. For the purpose of an example, we have chosen the Ballistic Missile Defense (BMD) problem—a real time highly data driven system in its own right. The BMD problem desires a data processing system architecture which is not only highly resilient to the threat but also provides for cost-effective growth to accommodate changing technology. With the PEPE architecture, the system growth can be achieved through the addition of PEPE elements—the system designer can incrementally purchase only those parallel processing elements needed.

THE BMD DATA PROCESSING SYSTEM ARCHITECTURE

The assumed data processing subsystem (DPS) used in this study is responsible for controlling the radar and interceptor subsystems to insure that a sufficient number of defended missiles survives an offensive ICBM attack. To accomplish this objective the DP subsystem performs twelve basic functions (Figure 1). These functions are categorized as either module level or unit level.

The module level functions are oriented toward operational control of multiple Defense Units operating asynchronously. The unit level functions perform the detection, discrimination and intercept of the threatening objects within the Unit's operational sector. Each Defense Unit contains a CDC 7700 data processor, a single, phased array radar and a complement of interceptors. BMD research is currently focused on only the Defense Unit level of the system, and the first eight basic functions.

The eight unit level functions constitute the application programs (AP) operating in each Defense Unit's CDC 7700. The AP contains an array of independent tasks which are either data or time enabled, and scheduled and dispatched by the real time operating system. Of the array of tasks, a subset constitutes the bulk of the real time resource load on the CDC 7700.

Normally, each of the applications tasks consists of multiple subroutine modules which, for testability and growth, are usually limited in size. Within each task is a task control routine which controls the execution sequence of the subroutines within the task, performs all global data base READ operations prior to task execution, and performs all global data base WRITE operations at the termination of the task execution. This access approach reduces the real time data management overhead.

The array of tasks can range in size from less than 2500 executable instructions* to well over 40000 executable instructions for each instance of data processed. In order to prevent any single task from monopolizing the system, a maximum task execution time can be imposed. This design constraint limits the number of data instances which can be processed per task execution. As a result, there are large variations from task-to-task, in the number of instances which could be processed per enablement. These variations when combined with variations in task execution priorities can result in queue buildups. In high threat load situations the queue buildups of some of the more data sensitive tasks can experience increases in execution wait times.

Under severe loads, system saturation will be characterized by queue overflows and corresponding increases in critical response time. The parallelism of PEPE provides the potential for handling such high load conditions by reducing the data processing system's sensitivity to the threat and increasing the system threshold of saturation.

PEPE ARCHITECTURAL OVERVIEW

The PEPE is a 32-bit super-computer of the SIMD and MIMD[1] class. The PEPE[2] (Figure 2) can contain up to 288 parallel processing elements. Each element is itself a multiprocessor containing three independently operating pro-

* Average Machine Language Instructions (MLI) derived from simulations of the applications process over a full engagement scenario.

cessors—an input correlation unit (CU) and an arithmetic output unit (AOU) operating at 5 MIPS*, and an arithmetic unit (AU) operating at 1 MIPS. Each processor in the multiprocessor element is under the control of a separate control unit which broadcasts decoded parallel instructions simultaneously to all 288 of the processing elements. The control units are themselves processors capable of executing sequential instructions while simultaneously issuing parallel instructions to the PEPE elements. The PEPE architecture, therefore, permits the concurrent execution of as many as six instruction streams; three serial instruction streams driving the control unit computations and three parallel instruction streams driving each of the three processors in the parallel elements.

In an operational environment in which PEPE is used as an adjunct processor to a large high speed Host, such as the CDC 7700, the PEPE will execute only highly data parallel computations. The PEPE will operate in a dedicated manner and will contain its own application load modules and operating system. The PEPE-Host communications will be used to only support process control and global data shared between separate processes executing on the Host and PEPE.

*Based on typical BMD computational instruction mixes.

PEPE IMPLEMENTATION—OBJECTIVE

The parallel architecture of PEPE, coupled with its inherent throughput potential offers an available low cost method for responding to the advances in threat technology. An implementation of PEPE which does not accompany a redesign of the software results in an inefficient use of the PEPE

LEVEL	NAME	DEFINITION
Unit Level	1. RRA/RS	Radar Return Assimilation and Radar Scheduling
	2. ODD	Object Detection and Designation
	3. OT	Object Track and Prediction
	4. IPP	Interceptor Planning and Prelaunch
	5. IC	Interceptor Guidance and Control
	6. URM	Unit (Radar and DPS) Resource Management
	7. DMC	Defense Module Inter-Communication
	8. OD	Object Discrimination
Module Level	9. MBM	Module Battle Management
	10. MSA	Module Status Assessment
	11. ESA	Environment Status Assessment
	12. CMP	Command Message Processing

Figure 1—Basic BMD functions

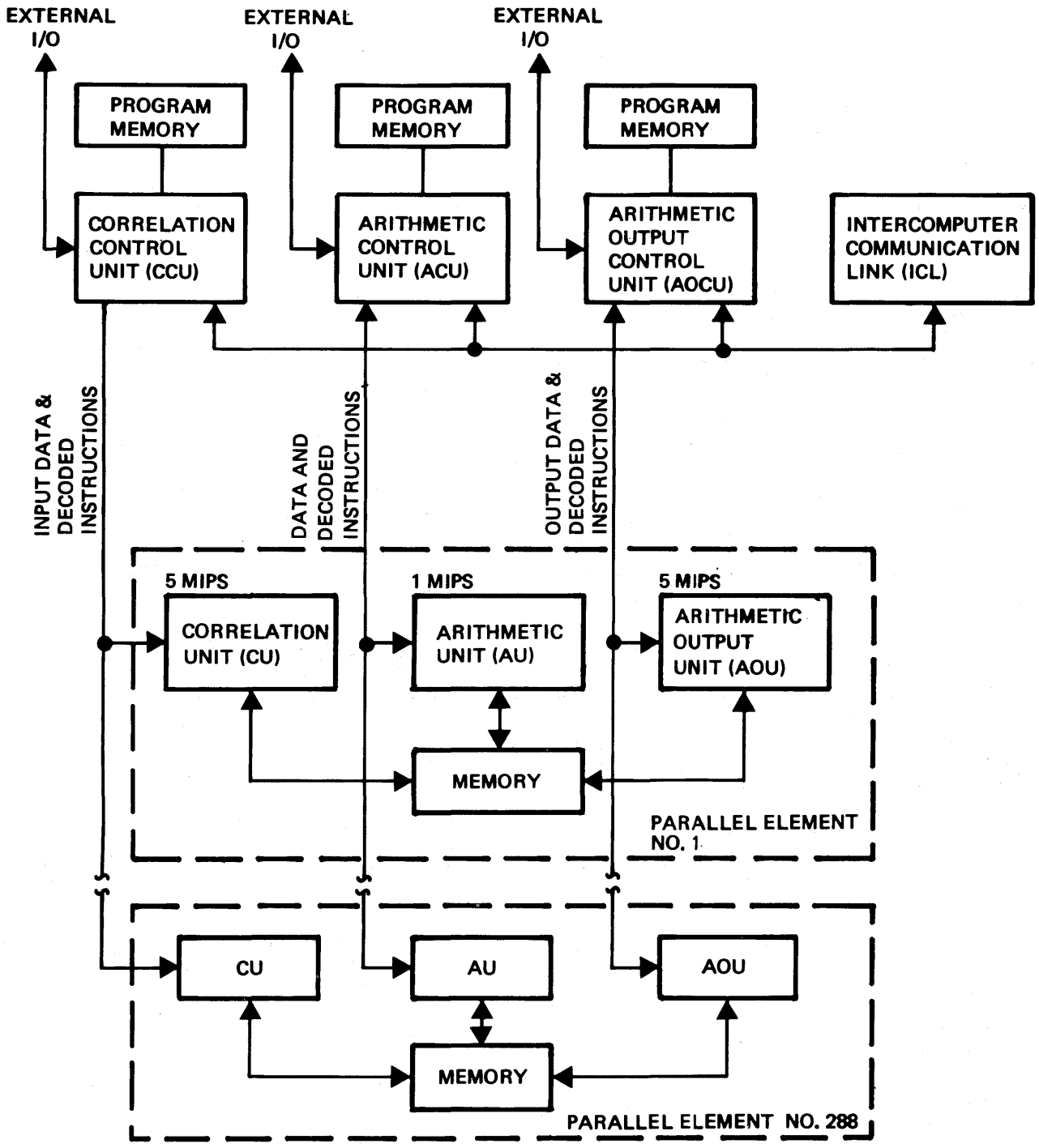


Figure 2—PEPE architecture overview

resources. However, the optimal allocation of a PEPE architecture to the dual CPU CDC 7700 architecture could involve substantial breakage.* The initial objective of the investigation was then to consider alternatives for low-cost (minimal breakage) implementations of PEPE. Therefore, any offload design requiring either a significant software redesign and/or a significant redesign of the PEPE-Host-Radar hardware interface was not considered. Similarly the optimization of the offloaded code was considered less critical than maintaining the original task software.

For each offload design considered both the application

*Breakage as related to the current DP system architecture (both software and hardware) was considered to include:

a. *New* Architecture (SW or HW) components.

b *Modification* of existing Architecture components.

c. *Disposing* of existing Architecture components.

task and operating system software were analyzed for breakage. Results of potential offload designs indicated that the application software breakage was highly sensitive to the number of application tasks included in the offload, while the indirect software breakage of the CDC 7700 operating system is essentially insensitive to the number of application tasks offloaded.

PEPE IMPLEMENTATION

Under the minimum breakage guidelines, there were two fundamental PEPE-Host architectures which were considered (Figure 3). One architecture (Figure 3B) employed PEPE as a radar interface preprocessor to the Host. As a preprocessor the PEPE would perform the radar interface function; correlation of radar returns and scheduling of radar pulses. However, designs of the radar interface software for centralized computers would execute extremely inefficiently on the PEPE, and the necessary software modification to improve the efficiency would be significant. The second architecture (Figure 3C) employed PEPE as an offload processor to the CDC 7700 in its current configuration. As an offload processor the PEPE would be dedicated to performing such highly data parallel operations as verification of detection returns, track initiate and track and discrimination processing.

Analysis of conventional applications programs was used as a basis for developing the offload options and selection of the preferred offload configuration. The analysis provided performance data which expressed resource requirements for: task computation (TASK), operating system support directly related to task execution (TOSC) (e.g., data base access), and operating system support indirectly related to task execution (e.g., external communication). All analysis was conducted within the high load periods of the engagement scenarios investigated.

PEPE IMPLEMENTATION—DESIGN APPROACH

Offload candidate identification

In an attempt to achieve the most cost-effective offload, the applications tasks were evaluated with respect to their current CDC 7700 resource requirements and their PEPE offload efficiency. Three aspects of the task resource requirements were studied,

(1) *Task Execution (TASK):* the resources (msec) required to execute the task instructions only.
(2) *OS Chargeable (TOSC):* the overhead directly associated with task execution (e.g., OS service requests: task load from secondary store, file read and write, task termination, etc.)
(3) *Other OS (TOSO):* the overhead not directly associated with task execution; not occurring during task execution (e.g., external I/O, interrupt handling, scheduling and dispatching).

The resource breakdown was used to provide an indication of each task's overhead sensitivity to the execution of instruction (TASK), handling of global data (TOSC), and external communication and polling loop priority (TOSO). Only those tasks which exhibited high resource requirements were considered as candidates for offload to PEPE. In addition, the general sensitivity of the tasks to the threat traffic was investigated by studying each in further detail during high load periods.

Those tasks which were rated as high resource users were then examined for inherent parallelism (both data and logic), as a basis for assessing the efficiency (breakage) with which a candidate task could be offloaded to the PEPE. Data parallelism was related directly to the task input data rates. With the minimum breakage guideline, algorithmic parallelism was not considered; instead, logic parallelism was evaluated at the subroutine level using the functional flow block diagrams. Tasks with a high degree of conditional branching along equal probability paths (Pb≈40-60 percent) would result in an inefficient use of PEPE resources. The lock-step architecture of PEPE can cause degraded performance in key processing tasks directly proportional to the number of branches in the offloaded processing logic.

A collection of six basic performance parameters was identified as providing the basis for assessing PEPE offload potential; they are,

(1) Total resource requirement
(2) Total number of instances processed
(3) Task resource distribution: TASK, TOSC, TOSO
(4) Total number of Task executions
(5) Peak data rates
(6) Task polling loop wait

Based on these performance parameters a set of five (5) performance criteria was established for paring down the application tasks to preferred offload candidates. These criteria were,

(1) high frequency of task execution
(2) high number of instances processed
(3) high instruction execution resources
(4) low data handling overhead
(5) high data queue buildup

In addition, the task structure was evaluated to identify those tasks with

(1) high probability logic paths ($Pb_n \geq 90$ percent)
(2) low offload instruction count with high frequency execution.

Using the above criteria the applications tasks were evaluated for their resource requirement. The results indicated that a small subset of the applications tasks accounted for over 90 percent of the total system resource requirements.

Analysis during a prolonged engagement interval provided performance profiles for the principal resource users. However, the analysis did little to indicate the sensitivity of the

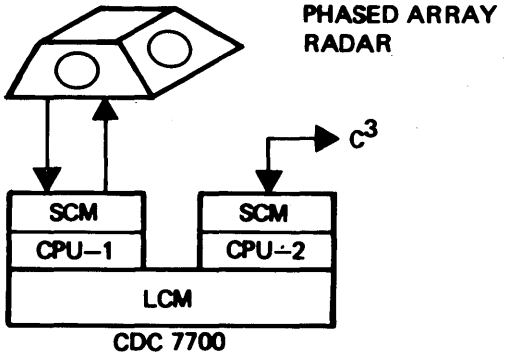


Figure 3A – CURRENT ARCHITECTURE

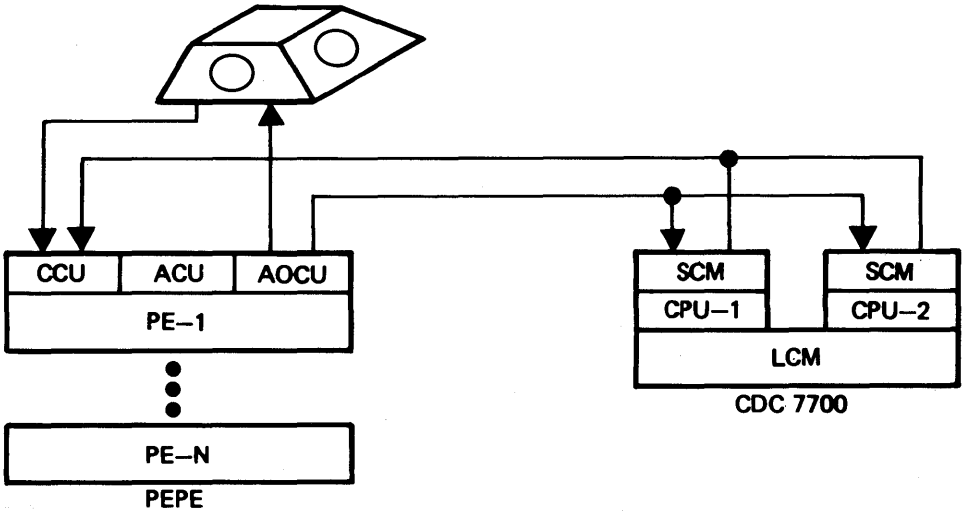


Figure 3B – PEPE PREPROCESSOR

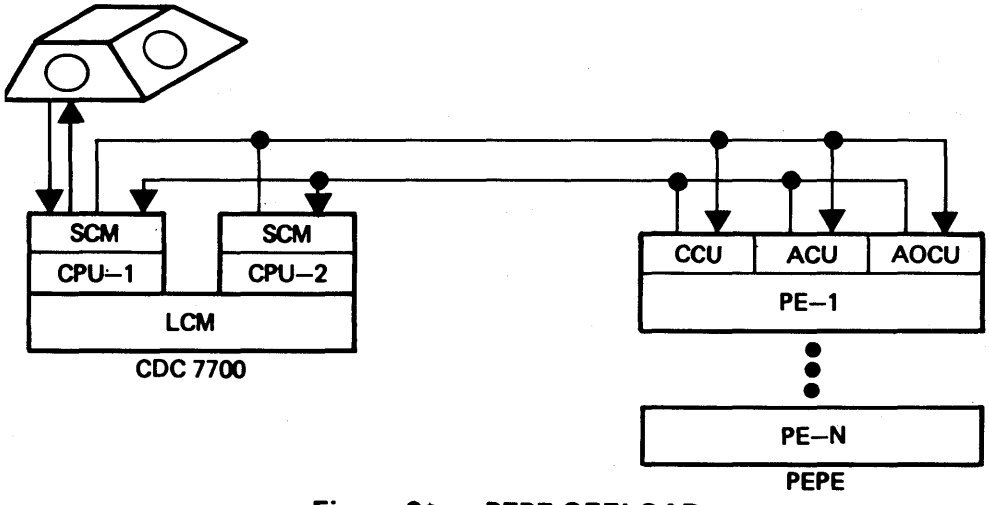


Figure 3C – PEPE OFFLOAD

Figure 3—BMD terminal defense architectures

tasks to threat traffic. Any offload configuration was restricted by the critically short processing response thresholds and limited offload storage available on PEPE. As a result, there was a need to develop criteria with which to assess the relative offload potential of each candidate task. The initial criterion was the resource offload percentage available due to each task. Tasks whose individual Host resource requirements failed to exceed at least five percent of the total applications program, were not considered as potential offload candidates. This test singled out five tasks as having the independent resource potential to justify offload.

The throughput capability of the PEPE relative to the Host, and the inherent parallelism of the PEPE architecture dictated two data related requirements. First, in order to exploit PEPE's parallelism large data loads were desired by the tasks for offload. Second, the cost to process a single instance should preferably be small when taking into account the potential expansion in processing times due to the relative PEPE-CDC clock cycles (100ns−27.5ns).

The first criterion strongly favored three of the tasks, with two of the tasks being marginal. One task did not meet the throughput requirement although the task's resource requirement is among the highest.

The second criterion of single instance processing costs strongly favors four tasks showing a high throughput requirement, with one task (interceptor control) exhibiting extremely high single instance processing costs. In addition, the stringent response requirement of interceptor control also makes it a poor selection for offload to PEPE.

A third performance consideration is the potential PEPE-Host communication overhead accompanying each task offload to PEPE. Large data communication overhead has negative impacts on both the PEPE-Host interface bandwidth requirements, and the responsiveness of the offloaded processing. Since the TOSC resource component is primarily influenced by data management costs associated with the global data, the TOSC component is a strong indicator of the potential PEPE-Host interface overhead. The ratio of TOSC to TASK resource distribution was then used to evaluate the inherent resource offload efficiency available with each offload candidate. The ratio ranged from approximately .25 to 1.14 and was found to remain fairly constant regardless of system loading.

Another consideration for evaluating task offload potential was the frequency of task execution. The execution rate of tasks when compared with their associated data rates provided another indication as to the efficiency with which tasks will execute on the PEPE. Each task execution is accompanied by a basic overhead cost (TOSC) consisting of loading the task module from large core memory (LCM) and the READ/WRITE of global data which support the task execution. This basic overhead is virtually insensitive to the number of data instances processed during a single task execution. If an execution time restriction were not imposed on the process designer, it would be much more efficient to have each task totally deplete the data queue on each execution. The parallel processing architecture of PEPE provides a means of processing a much larger number of data instances per task execution while still maintaining an execution time design restriction.

Offload design—General approach

Minimization of the impact of any offload design required the development of a technique for restructuring the current application software to support offload to PEPE. The technique which we developed was based on partitioning the applications tasks along their current subroutine structure. In addition, the table driven design for the current operating system allowed the offload to be implemented with no impact to the OS software.

The partitioning approach to PEPE offload involved the segmentation of each offload task into three components (Figure 4) operating as three tasks. Two new data files were defined which serve as data enablement queues for the second and third segments. The first and third segments of the task would continue to reside in the Host computer. These segments provide all access to the global data base, and interface with the remainder of the application process. The second segment would reside on the PEPE. One objective of the partitioning was to maintain the total cost of executing the two segments residing on the Host≤the original cost of the task prior to offload.

In general the segmentation will be performed across loosely coupled routine interfaces in the task logic, and produce a segmentation in which the largest segment of the task is placed on the PEPE.

The initial segment will continue to execute in the Host as a data enabled task. The primary purpose of this Host segment will be to format and write into the newly defined PEPE input queue all data necessary to support the second task segment residing in the PEPE. The data will typically include the data instances, such as radar returns, and dynamic data which is needed to support the execution of the PEPE segment.

The second segment would be offloaded to the PEPE and allocated to either the CCU, ACU, or AOCU. The PEPE will contain the complete load module of the second segment loaded into the program memories of the CCU, ACU, or AOCU. Any fixed point arithmetic or logical operations of the offloaded segment, such as unpacking/packing operations, will generally be performed by the CCU or AOCU. The ACU will perform all floating point computations. In

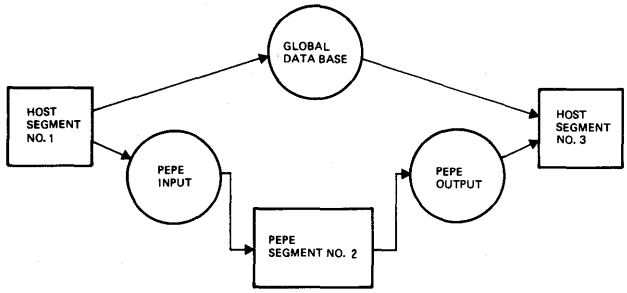


Figure 4—Task offload segmentation technique

addition, new PEPE software must be developed for the CCU and AOCU to support data manipulation within the PEPE, such as loading the PEPE elements prior to parallel computations and retrieving the processed data from the elements for retransmission back to the Host following computations. In addition, for data management operations such as large file searches/correlations, new software must be incorporated into the offloaded segment which exploits the PEPE's associative capabilities. The PEPE Fortran (PFOR) possesses instructions specifically designed to support associative search operations within the PEPE elements. Typically operations such as object redundancy checks and nuclear interference prediction which could be extremely costly when implemented on the Host, will be executed with only a couple of associative search instructions on PEPE.

Alternative offload designs

One of the major applications tasks consistently displayed performance characteristics indicative of efficient PEPE offload. That task performed the object track function. The keys to the offload desirability of the track task were four operational characteristics,

- *high task execution frequency* with highest execution rates occurring during periods of highest threat traffic.
- *high task throughput requirement* with highest data loads (75 percent) occurring during periods of highest threat traffic.
- *high CDC resource requirement* with largest load (80 percent) occurring during periods of highest threat traffic.
- *low task execution overhead* (30 percent) throughout the operational period.

Because of the task's offload desirability it was used to form the foundation of two alternative offload designs.

In addition, the task has a moderate per instance execution cost, a highly favorable characteristic for tasks belonging to short response threads.

Using the track task as a basis, two multi-task offload designs were developed. The first design was directed at significantly reducing the total Host resources requirement. The second design was directed at substantially improving the responsiveness of key processing threads in high threat traffic situations. The first design incorporated the detection and track initiate and discrimination processing tasks with the track task into an offload module for the PEPE. This design provided the largest offload benefit in the early portion of the engagement. In addition, the detection and track initiate tasks are on processing threads with less stringent response requirements than the track thread which reduces the impact of resource contention. The second offload incorporated the interceptor control task with the track task into an offload module for the PEPE. The interceptor control task was a high resource user with an expensive per instance execution cost. In addition, the interceptor control task was on the most stringent response processing thread in the applications program.

A comparison of the two integrated offload designs indicated the first design to be far more preferable. The offload of a sizable portion of the Host resources, also had a significant indirect effect on improving the responsiveness of the threads remaining on the Host. The offload to PEPE of tasks within high response processing threads had less effect on improving system responsiveness than the first offload design. This was attributed to the fact that, (1) less absolute resources were offloaded from the Host to PEPE, and (2) the offload of tasks with equal response priority created resource contention on the PEPE. An important point of both offload designs was the ability of PEPE to substantially improve the performance and predictability of both system throughput and responsiveness.

PEPE IMPLEMENTATION—ANALYSIS

Offload design—S/W portability analysis

Parallel Fortran (PFOR)

Parallel Fortran[3] is a high level, procedural extension to USA Standard Fortran and provides a powerful and efficient set of language extensions for the associative parallel processing capabilities of PEPE. Through the use of the PFOR statements, the user has the capability of simultaneous execution of instructions in all n PEPE elements or a selected subset of the n elements. The user can nest the levels of element activity through PFOR.

In order to identify specific variables which are allocated to PEPE element memory, PFOR has parallel type declaration statements.[4] A set of parallel "system" functions to handle type conversion, square root, and special mathematical functions on parallel arguments and a set of powerful statements are included in PFOR for the subsetting of the current set of active elements. This set includes statements of the form WHERE, CONVERGE, parallel IF, and parallel DO. In addition, a statement is provided for moving data between element memories.

CDC 7700 S/W conversion

In order to determine the amount of code breakage and time required to convert existing code in FORTRAN to PFOR (Parallel FORTRAN), typical code representative of the object track task was converted. The routines were selected as being strongly indicative of the types of code which were contained in the PEPE segments of the offload design.

Some of the code was highly logic oriented and designed to control the hi-fidelity Kalman filter for the track task. The other code contained mostly arithmetic operations and little branching.

There are two options which can be considered when placing the selected subroutines and their associated data base on PEPE. Option one permits only one object maintained per PEPE element and requires a larger set of PEPE

CDC 7700/FTN

```
      SUBROUTINE XYZ

      COMMON/D3T0000/A3T00 (132)
      INTEGER ZDBQN, . . .
      EQUIVALENCE (A3T00(1), ZDBQN)
      IF (Z3DØF .EQ.0) GO TO 10
      Z3BPF = 1
10    CONTINUE

      IF (Z3DØF.EQ.0) GO TO 70
      ZNC (1, Z3ØDV) = ZNC (1, Z3ØDV) +
      FLØAT (ZNCNL (Z3BQV))
      GO TO 80
70    ZNC (1, Z3ØDV) = ZNC (1, Z3ØDV) +
      FLØAT (ZNCNL (Z3BQV)) + 1.
80    CONTINUE
```

PEPE/PFOR (OPTION 1)

```
      SUBROUTINE XYZ, ACU

      PAR COMMON/D3T0000/ZDBQN, . . .
      PAR INTEGER ZDBNQ

      WHERE (Z3DØF.NE.0) 10
      Z3BPF = 1
10    CONTINUE

      WHERE (Z3DØF.NE.0) 70
70    ZNC (1) = ZNC (1) + PFLØAT (ZNCNL(Z3BQV))

      WHERE (Z3DØF.EQ.0) 80
      ZNC (1) = ZNC (1) + PFLØAT (ZNCNL(Z3BQV)) + 1.
80    CONTINUE
```

Figure 5—Relative code comparison

elements. In converting the code for this option, the user must change the doubly subscripted variables currently used throughout the software (e.g., object state estimate) into singly subscripted variables. The second subscript associated with the instance number (e.g., object identification) would be implied by mapping each instance into a unique PEPE element. This option results in significant code modification, however, the code conversion would be simple, straightforward and could be automated.

Option two would permit more than one object maintained per PEPE element and can significantly reduce the size of the PEPE element array required. This option requires doubly subscripted variables to identify the current instance being processed in an active element. This option would result in much less code breakage than option one but would require much larger data storage per element. Once the user develops some familiarity with PFOR a set of existing routines can be transformed into PFOR in relatively short time. As an illustration, Figure 5 provides an example of a logical segment of the subroutine and the FORTRAN code changes required to convert it to PFOR. In practically all cases of code conversion, the code which required modification was directly associated with data management and manipulation. The actual scientific computations written in FORTRAN were directly transportable to the PEPE. For the three subroutines that were selected, approximately 76 percent of the executable statements would require modification for option one and 4 percent for option two.

In the offload design, the current CDC 7700 data base design is not effective when directly installed on PEPE. In each offload design the subroutines executing on the PEPE require only small segments of the data contained in current common blocks. To conserve element memory we elected to restructure the common blocks within the PEPE. The restructuring involves eliminating unused variables and redefining the nonexecutable statements which define these variables. Since the current set of declarative statements (i.e., INTEGER, REAL, EQUIVALENCE) have to be converted to PFOR (i.e., PAR INTEGER, PAR REAL) there is total breakage for these nonexecutable statements.

It is felt that other applications which are suited to the type of parallel processing offered by PEPE could just as easily be converted to PFOR. In some cases the code conversion could even be automated. Applications which would require movement of data between elements may require a longer time to convert to efficient PFOR, but once the user becomes familiar with PFOR, the code conversion is straightforward.

Offload design—performance impact

Analysis indicates that the first offload design exhibits a significant overall potential for reducing the load on the Host computer. Over the complete engagement the offload design provides a net reduction in the average Host resource requirement of the applications program by over 25 percent. This is accomplished by a 54 percent reduction in the resource requirements originally required by the offloaded tasks. The offload design provides the most significant reductions during the major peak load periods of the engagement; specifically early during initial acquisition of the threat and later during track and discrimination prior to intercept

commit of the threat. The early load is caused by high Host resource requirements imposed by the track initiate loads. The later Host load is caused by high Host resource requirements imposed by the object track and discrimination processing. The offload to the PEPE provides a significant reduction in the Host loads during both the early (~28 percent) and later (~23 percent) portions of the engagement.

An analysis of the PEPE under this offload design indicated that both storage and utilization requirements were well within available limits. The program storage requirements for each of the three control units were all less than 35 percent of available storage. Data storage requirements were higher and reached 84 percent of that available in the CCU, and 65 percent of that available in the PEPE element memories. Evaluation of the PEPE resource requirement during the engagement indicated that none of the control units exceeded 80 percent utilization. The average utilization of these units was substantially less, never exceeding 60 percent (ACU), indicating resources are available for additional applications tasks to be offloaded.

The size of the Host resource offload is directly related to the traffic load on the system and as a result varies during the engagement. A major benefit of the PEPE offload design is the reduction in sensitivity of the PEPE-Host system resource requirement to increases and unpredictable perturbations in data (threat) traffic patterns.

The impact of resource offloads on the system responsiveness was also investigated. In the original Host environment there were two principal components of system response, the first component being that of execution wait times, which includes the time from enablement to dispatch. The second component is the actual execution time once dispatched. In high load periods of resource contention, the task wait times become a major component of the overall system responsiveness for the Host alone system. With the offload to the PEPE, a third resource component must be considered, that of PEPE-Host communication. In the offload design, the contention for resources is reduced, and the execution wait times are decreased. In addition, the execution wait times tend to be more stable with the PEPE adjunct than with the Host alone.

When combining the effects of reduced task execution wait time, increased task throughput and the addition of the communication delays, the results indicate that the PEPE-Host system provides improved system responsiveness. This responsiveness is substantially improved during periods of high load conditions.

One of the major offload issues addressed the impact that the PEPE adjunct, and its associated PEPE-Host communication, had on the Host Large Core Memory (LCM) utilization overhead. Analysis indicated that the LCM overhead actually decreased with the offload. Although the Host LCM overhead necessary to support real time input/output (RTIO) increased with the addition of the PEPE, the reduction in the overhead associated with management of some of the larger data files and load modules, now resident in the PEPE, more than compensated for the RTIO overhead increase.

SUMMARY

THE PEPE provides a highly cost-effective method of improving the data handling capacity of a real time system developed around a single large-scale Host. Analysis of conventional terminal defense systems for BMD indicates that the addition of a PEPE adjunct can provide a means to substantially offload the Host computer. As a result of the Host offload, an improvement can be achieved in the capacity and predictability of both the system's throughput and responsiveness.

In addition, the implementation of PEPE can be accomplished with only a minimum breakage to software developed for the Host. In most cases the software breakage is repetitive but not complex and can be automated without significant cost. For the specific offload design investigated, the PEPE storage and throughput requirements were well within limitations and PEPE had the resources to accommodate additional offloads. Based on the current design the apparent threshold in PEPE offload will be the availability of data storage. Analysis indicated that through a redesign of some of the larger common areas the data storage requirements on the PEPE can be significantly reduced.

Further analysis of the offload design indicated that the PEPE adjunct provided several direct performance improvements for the data processing system,

(1) increased data throughput for tasks offloaded to the PEPE,
(2) reduced execution delays for entire application program,
(3) reduced data management overhead,
(4) reduced system sensitivity to data traffic,
(5) improved predictability of the system's real time resource demand.

It is felt that these performance improvements can be achieved as well with non-BMD applications which possess the parallelism which is suited for the PEPE. Once the PEPE user becomes familiar with the operation and the computing power of the PEPE, suitable applications can be mapped onto PEPE in a simple and straightforward manner.

CONCLUSIONS

Results of this PEPE application case study have general applicability to most data driven, real time systems. The inherent parallelism and programmability of the PEPE make it an extremely attractive mechanism for achieving significant improvements in a centralized systems data handling capability.

The implementation of PEPE as a Host adjunct allows the offload of resources to the PEPE. This offload can produce a substantial improvment in the data processing system's throughput and response. The improvement is effected in both the system's capacity and performance predictability to data loading.

REFERENCES

1. Flynn, M. J., "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, September 1972, pp. 948-960.
2. Cornell, J. A., "PEPE Architecture—Present and Future," National Computer Conference, June 1978, Anaheim, California.
3. System Development Corp., "Preliminary Users Manual for the PFOR Language Translation System," SDC, TM-HU-046/400/00, 20 August 1973.
4. Cornell, J. A., "PEPE Application and Support Software," *WESCON* 1972, September 1972, pp. 1/3-1 to 1/3-3.

Special computer architectures for pattern recognition and image processing—An overview

by K. S. FU
Purdue University
West Lafayette, Indiana

INTRODUCTION

It has been recognized for a long time that a conventional sequential processor is inefficient for operations on pictorial data where relatively simple operations need to be performed on a large number of data elements (pixels).

Though many parallel processing architectures for picture processing have been proposed in the past, very few have actually been implemented due to the costs involved. With LSI technology, it is becoming possible to realize parallel architectures at a modest cost.

In the following we review some of the proposed architectures for pattern recognition and image processing.

PREVIOUS WORK AND COMMENTS

Previous work in the field of special purpose computer architecture for image processing basically falls into two categories: bit-plane processing and distributed processing. The bit-plane processing approach performs the arithmetic computation on the image points which are stored in Boolean bit-plane. For example, if the image has eight grey levels, then the image points are stored in three Boolean planes. The bit-plane processing approach tends to have a large number of processors which perform Boolean operations. The distributed computing approach utilizes processors which have powerful computation capability. These processors are designated by microprogram or hardware to execute certain specific tasks. Thus, this configuration forms a distributed computing architecture. In this section we will briefly illustrate each special purpose computer, concentrating on the special features of each computer for image processing and pattern recognition.

Unger[1] realized as early as 1958 that a sequential processor is inefficient for problems involving spatial relations and proposed an array computer (Figure 1). The system is made up of central control (Master Control) controlling a large rectangular array of identical modules interconnected as shown. The master control performs all instruction fetches and decoding and broadcasts the decoded instruction to all the modules in the array.

Each module consists of a 1 bit accumulator and a small number of memory elements (a,b,c . . .). Inputs to each module are from the M.C. and the accumulators of the four neighbors (above, below, left, right). Each module can perform the following basic operations:

1. LOGICAL OR (AND)
   The accumulator is ORed (ANDed) with specified memory register (a,b,c . . .) and the inputs from the specified neighbors. The results are either stored in the accumulator or one of the memory registers.
2. LOGICAL NOT
3. LOAD (STORE)
4. SHIFT (ROTATE) LEFT (RIGHT,UP,DOWN)
   The entire image is shifted (rotated) in the direction specified.
5. LINK/EXPAND H (V,UP,DN)

These operators are used to determine the connectivity in the specified direction. Thus, for example, it is possible to determine all the cells in a given field which are connected to a specified cell through a chain of horizontally linked ones (Figure 2). Using these commands, many interesting image processing operations can be performed. But the actual programming of algorithms is nontrivial.

Unger's architecture has influenced that of many later researchers. Kruse's PPM, the CLIP array, as well as the ILLIAC III computer were inspired by it.

ILLIAC III

The Illinois Pattern Recognition Computer, ILLIAC III,[2] was designed for automatic scanning and analysis of massive amounts of relatively homogeneous visual data, in particular, bubble chamber negatives. The computation is performed by three units, the pattern articulation unit (PAU), the taxicrinic unit, and the arithmetic unit. The pattern articulation unit performs local preprocessing on the digitized raster, such as track thinning, gap filling, line element recognition, etc. The logic design of the digital computer has been optimized for the idealization of the input image to a line drawing. Nodes representing end points, bends, points of intersection are labeled in parallel by appropriate programs. The abstract graph describing the interconnection of

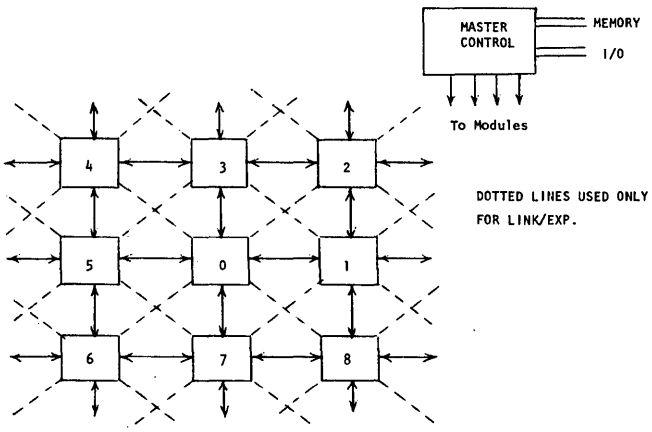


Figure 1—Module interconnection pattern for Unger's machine

labeled nodes is then extracted as a list structure, which comprises the normal output of the processing element (stalactite). The pattern articulation unit consists of an iterative array of 1024 identical stalactites (32×32). A schematic diagram of the stalactite is shown in Figure 3. The iterative array of the stalactite can be connected in, either rectangularly or hexagonally, at the programmer's option. Each stalactite can accept an input from any of the eight neighboring stalactites, S0, S1, . . . , and S7 shown in Figure 3 (rectangular) or six neighboring stalactites (hexagonal) in the plane and from itself, M shown in Figure 3. The input signals are logically "OR"ed, optionally complemented, and stored in one (or more) of the nine planes. Communication with the supplementary planes in the core buffer is through the "M" plane, which serves as the buffer register of this memory. For output, the output of any selected set of planes can be logically "AND"ed and passed on to neighboring stalactites. With a special signal, the stalactite allows an input signal to pass through it directly, without interim storage. It is this feature which allows path-building within the machine. The set of 32×32 stalactites processes the pixels of a 32×32 pixel image window simultaneously. The whole image is to be processed by one image window (32×32) after another sequentially by the set of 32×32 stalactites. The taxicrinic unit assembles the graphs, which are outputs of the pattern articulation unit, into coherent list structures and categorizes these graphs to complete the visual recognition function. The arithmetic unit is designed for executing mathematical analysis, such as stereoreconstruction, statistical summarization, etc. involved in processing pictorial data. The block diagram of ILLIAC III is illustrated in Figure 4.

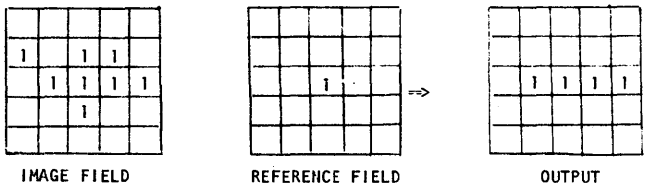


Figure 2—Link/expand operator

The generic stalactite contains 10 one-bit registers (flip flops). Since asynchronous logic is employed, it is required that no flip flop be both read from and written into during the same instruction to avoid races.

The pattern articulation instruction code is of 40 bits length, made up of 4 groups.

Flow group

C	G	CI	F	B	U	D	N2	N1	N0
1									10

C, CI are used to control the outgoing and incoming signal complementation respectively.

G=1 indicates that at least one input gate is opened.

F=1 indicates a flash through operation, i.e., the input signal is transferred directly to the output bypassing the stalactite.

B=1 indicates a bubbling operation.

In bubbling the values of the nearest neighbors are read. Using a sort, the bubble register (flip flop 1,2,. . . ,8) 15 ordered such that zeros are at the top and the ones at the bottom. The number of ones is counted and the cell is marked if it exceeds a threshold. This operation is used extensively in line thinning and other preprocessing operations. U specifies bubble up, D specifies bubble down, N2, N1, N0 specify the number of iterations needed for bubbling.

FF Read control group

M	0	1	2	3	4	5	6	7	8
1									10

This field is used to select the FF's to be read out.

Input gate control group

C	0	1	2	3	4	5	6	7	8
1									10

C=0 All inputs signals enabled C=1 Only specified signals enabled

FF Load control group

M	0	1	2	3	4	5	6	7	8
1									10

This field specifies the FF's to be loaded.

Mark command

This command is of the form MARK(Z). It sets the M register of the cell at coordinate Z=(X,Y).

LIST Command

List serially compiles a list of all coordinates Z which have a zero in the M FF. Coordinate Z of the next point found is transferred to the LIST register automatically on storage of the previous contents of the register.

The use of the above two instructions is illustrated below. A sparsely populated image is complemented and loaded into M reg of the PAU array. The black points in the image correspond to a 0 in the M plane. A first black point is identified by the LIST command. The coordinate obtained is used to mark the corresponding M FF (and thus erase the point). Now the black points connected to this point are in

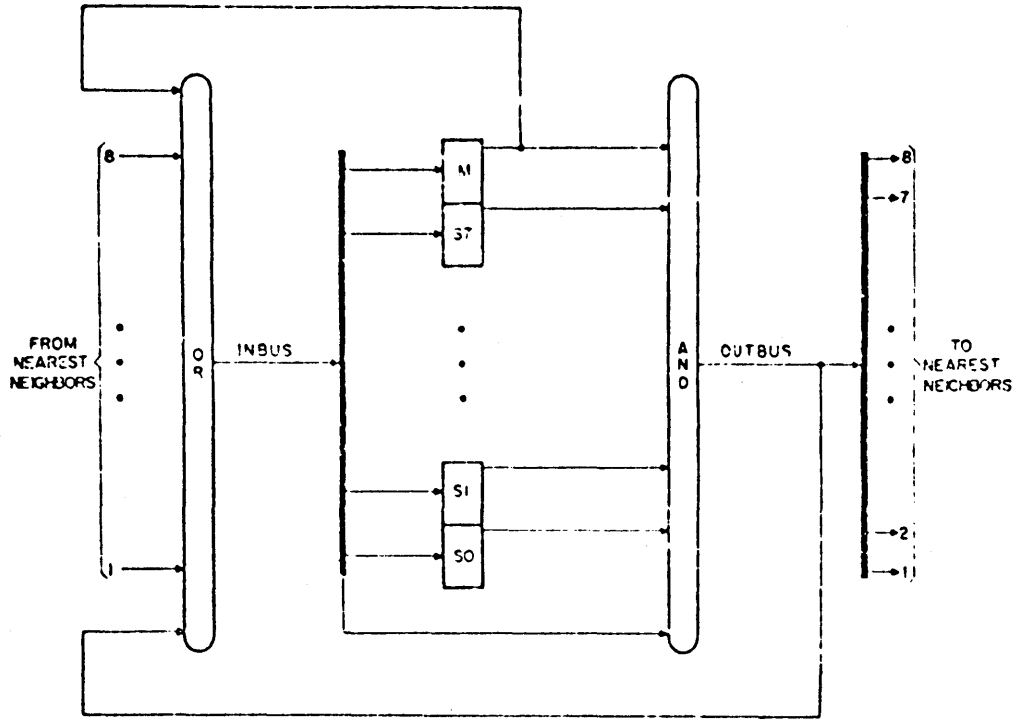


Figure 3—Schematic diagram of stalactite of pattern articulation unit in ILLIAC III computer

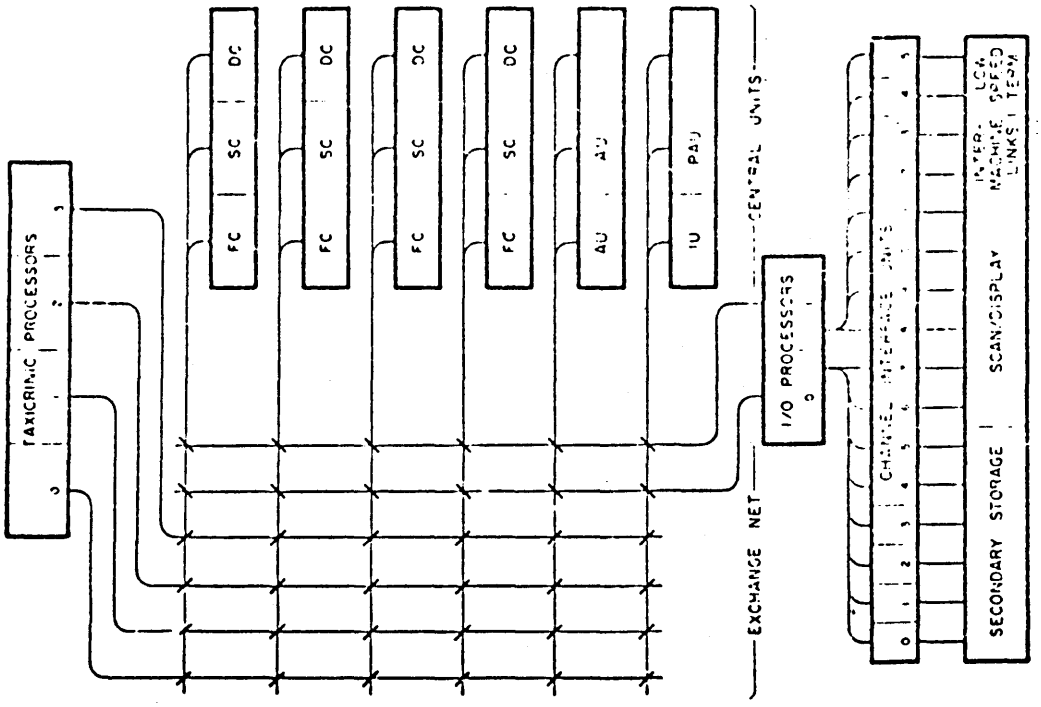


Figure 4—Block diagram of Illinois pattern recognition computer ILLIAC III

turn listed using Flash through. This technique builds a paired list of nodes, i.e., the branches of the associated graph.

One special feature of ILLIAC III for image processing is the use of an auxiliary memory. The auxiliary memory is used for the processing of images frame to frame, and for retaining intermediate partial results. The necessity of storing intermediate results of the iterative array stems directly from the manner of executing homogeneous logic transformations in the processor. The complexity of the iterative array can be greatly simplified if various digital filtering operations can be performed serially storing the intermediate results (32×32=1024 bit words). For example, local track segment orientation can be designated as being predominantly horizontal, vertical, left-diagonal, or right-diagonal in four serial tests, whereas simultaneous identification requires approximately four times as much hardware.[2] Unfortunately, ILLIAC III has never been completely built since 1967.

The ILLIAC III represents a pioneering effort in designing a special processor for a specific image processing task. The processing capability is, however, limited due to the technology available at the time. The complexity of the system can be imagined from the fact that each stalactite occupies one circuit board, and 1024 boards make up the PAU array. Recent advances in LSI technology make building of such arrays economically viable as demonstrated in such systems as the CLIP array described later. By using more complex logic in the modules, it is possible to realize truly general image processing capability.

CLIP Series

Digital Parallel Processors (DPP's),[3-8] using cellular logic, are real-time machines in which the action resulting from a program statement is simultaneous on all the points of the array. The action may be symmetrical or directional and the tessellations of various types, the most common being the square and the hexagonal ones. An example of this kind of machine is the parallel cellular logic image processor, CLIP 3.[6,7] The CLIP 3 is comprised of an array of 192 cells arranged in a block of dimensions 16 cells (vertically) by 12 cells (horizontally). The array interconnection pattern can be either square or hexagonal. The required architecture is determined by one bit in each instruction word and is therefore under the control of the programmer. The block diagram of the cell logic of CLIP 3 is shown in Figure 5. The Boolean processor can perform, under program control, two independent Boolean functions from its two inputs A and P. D is one output and can be regarded as the processed pattern bit corresponding to the cell. The other output, N, fans out to the adjacent cells. The N inputs from neighboring cells (G_1, G_2, . . . , and G_8) are summed (Σ) and compared with a threshold (f). The programmer is allowed, for each PROCESS instruction, to select several inputs to be put into the summing unit and he can also choose the threshold value. The summed and thresholded output, T, is then "OR"ed with the second pattern input to form the input, P, to the Boolean processor. D outputs are addressed into any one of 16 single bits of store; buffers A and B are loaded from addressed location in the same store. The CLIP 3 was actually built.

One drawback of both the ILLIAC III and CLIP 3 is that the "edges" caused by moving image windows, (in ILLIAC III 32×32 windows and 16×12 windows in CLIP 3)) are not taken care of by the computer. The computing power of the CLIP 3 is obviously limited because of the fixed small number of (Boolean operator) cells. (16×12=192 cells).

In order to process larger images, CLIP 3 has been interfaced to a television camera through the hardwired scanning unit. This is called a hybrid CLIP 3 array[8] which is shown schematically in Figure 6. This unit scans the 192 cell CLIP 3 array across the 96 by 96 cell data-field and provides edge stores to handle the propagation signals which cross between

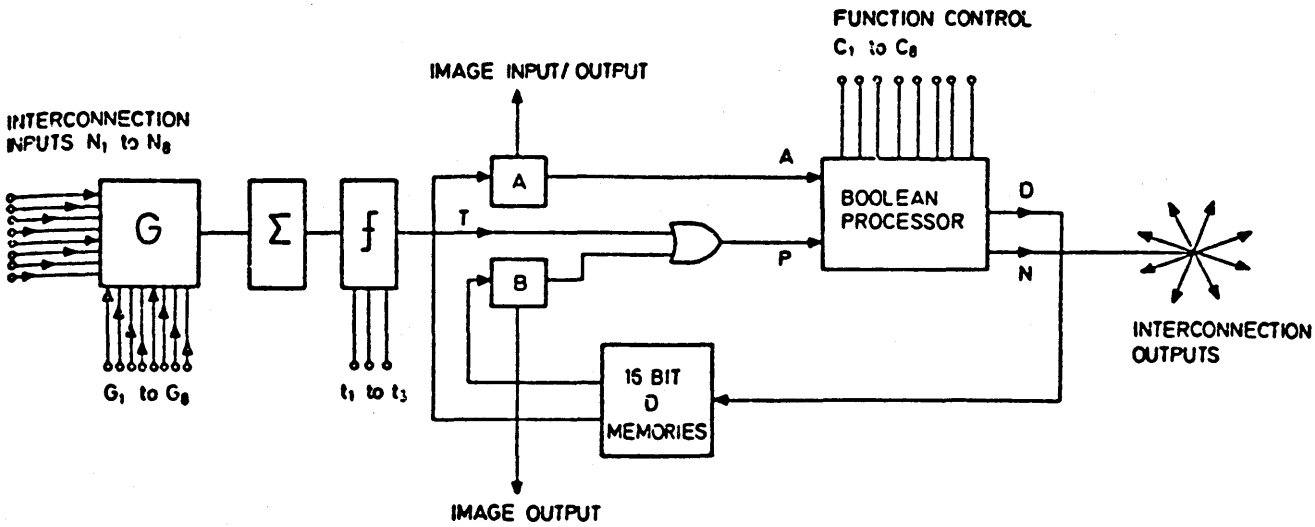


Figure 5—CLIP 3 cell logic

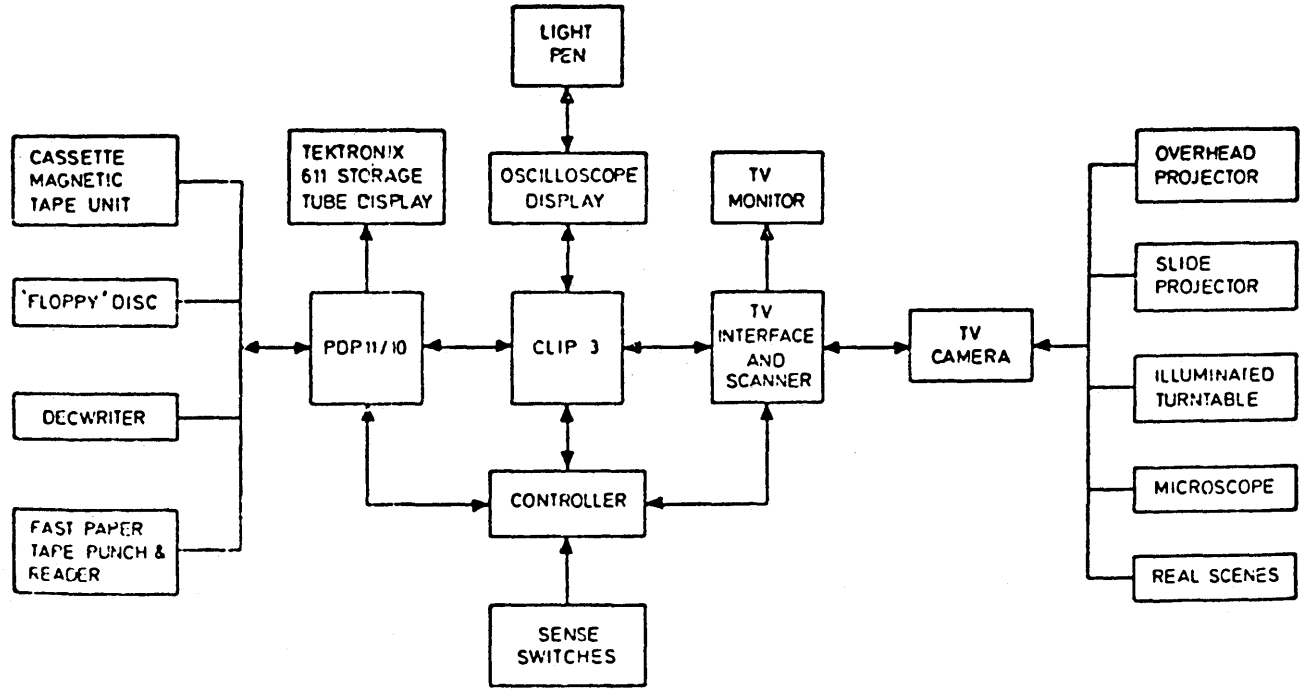


Figure 6—Schematical diagram of hybrid CLIP 3 system

adjacent sectors. The complete system is interfaced to a PDP 11/10 computer which serves to extend the available data and instruction storage and also provides program editing and assembling facilities.

CLIP 4[8] is the Large Scale Integrated circuit (LSI) version of CLIP 3 with some small changes in the cell design. The CLIP 4 uses N-MOS (N type Metal Oxide Silicon) LSI to incorporate eight processor cells in a four by two block onto one chip. The block diagram of CLIP 4 cell logic is shown in Figure 7. The "D" store has been increased from the 8

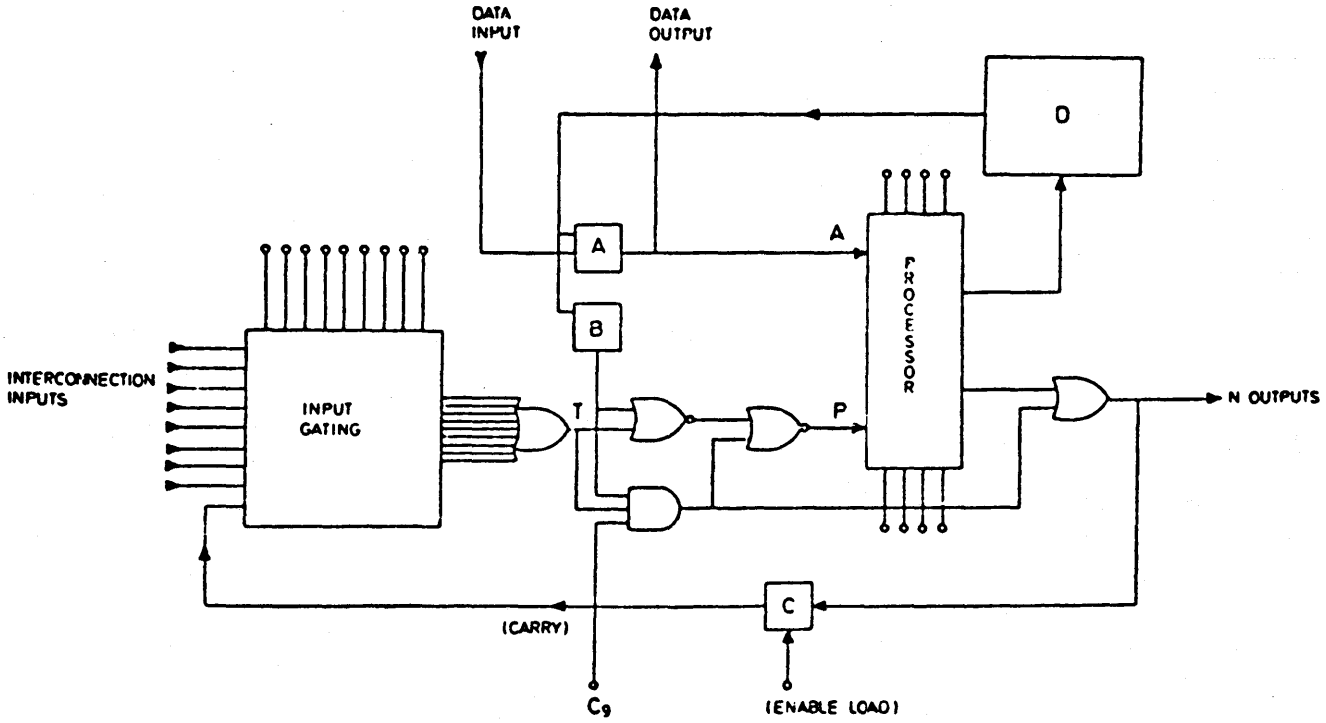


Figure 7—CLIP 4 cell logic diagram

bits of CLIP 3 to 32 bits which allow the processing of 32 grey levels. The interconnecting threshold gate has been replaced by an "OR" gate, and a few extra gates and an additional buffer store have been included to provide automatic carry-in arithmetic operations.

The experimental CLIP 4 system consists of 16×12 identical cells. The array interconnection pattern can be either square or hexagonal with six neighbors. The desired architecture is programmed by a bit in the instruction. The processor in the CLIP 4 is a Boolean processor capable of implementing any Boolean function of inputs A,P. Thus,

$$D=b2(P,A),\ N=b3(P,A)$$

The actual functions b2,b3 are determined by the control lines c1, c2, . . ., c4 and c5, c6, . . . , c8 respectively. (All 16 possible Boolean functions are implemented for D and N).

The B register is primarily intended for comparing two images. D is a 32 bit random access storage in each module. The input gating is used to enable/disable inputs from neighboring modules. This permits selective propagation of the interconnection signal in desired directions within the array. Thus,

$$T=G1.I1+G2.I2+. . .+G8.I8+G9).Carryin$$

Carryin is a carry bit used to perform bit serial arithmetic.

The normal mode of operation of the array is as follows. The input image is digitized by a flying spot scanner and loaded into the A register of the array modules. Once the A registers are loaded with the desired image we can perform operations on it. The results may be stored in the D memory. A CRT is used to display the output.

Instruction set

The array is controlled by a sequence of instruction words stored in external memory. Instructions which do not involve signal propagation through the entire array are executed in 1 microsecond. Delays for operations involving propagation depend on the array size as well as the image being processed.

There are basically three types of instructions recognized by the assembler:

1. LOAD
   LD (A addr), (B addr), (D addr)

The instruction loads A and B registers from the locations specified in the D memory. The D address is used by the subsequent PROCESS instructions as the destination address for the operations.

2. PROCESS
   PR (Direction list) b3, b2, E S

The 'Direction list' specifies which neighbor inputs are to be enabled (G1, . . . , G8). b3 is the Boolean function for the interconnection output, b2 is the Boolean function for the image output, which is stored in the D memory location specified by the previous LOAD instruction. E is a binary number (0 or 1) specifying the inputs at the array edges and S indicates whether square or hexagonal interconnection pattern is desired.

3. BRANCH
   This is a conventional program branch instruction.

Analysis of CLIP operation

Each cell in the CLIP array may be partitioned into three parts:

1. Next state logic
2. Propagation Logic
3. Memory

The interconnect signal propagation logic is combinational and there is no delay between elements. Thus, during each clock period, signals propagate through the entire array, the propagation depending on the function b3 (set by c5, . . . , c8).

For simplicity in the following analysis, we assume that register B=0, c9=0. The operation is best studied in terms of interconnect signal propagation as below.

1. ZERO ORDER
   Here D=0 or 1 independent of interconnection signal. Thus this is a trivial case.
2. FIRST ORDER
   In this case D=A or D=NOT.A independent of N.
3. SECOND ORDER
   D=b2(A,T), N=b3(A)

Note that T is the input from the neighbors. Also the interconnect signal N depends only on the module state A and not on the interconnection signal input T. Thus the output of each module depends only on the current state of the module and that of its immediate neighbors (determined by G1, . . . G8). Hence the maximum delay is only 2+t where t is the delay per module.

4. THIRD ORDER
   These functions result when N depends both on the input T as well as the state of A. All stable functions in this category are third order functions. Since interconnect signal propagation occurs through the entire array the execution of these instructions is slow. These are the most interesting functions that may be executed on the CLIP array.
5. FOURTH ORDER
   These correspond to those operations for which the output is unpredictable due to logical races within the array. An example is the operation

$$N=A.T$$

Two consecutive 1 cells can result in a latch up with unpredictable output.

Comparing the performance of CLIP 3 and CLIP 4, the processing speed of the CLIP 4 cell is slower than that of CLIP 3 by at least a factor of five. A single operation which required a pair of instructions (LOAD and PROCESS) in CLIP 3, taking 2 μsec, is expected to require about 10 μs in CLIP 4. Propagation from cell to cell took 0.1 μs in CLIP 3 and may take 1 μs in CLIP 4.[8] However, the processing speed of the overall system of the CLIP 4 is improved by the larger array of processors of CLIP 4 (96×96 cells).

PPM and PICAP

The Parallel Picture Processing Machine (PPM)[9] is a special processor which is connected to and controlled by a conventional computer. The block diagram of PPM is shown in Figure 8(a). The PPM consists of the following principal parts: the processing unit, a set of nine general purpose picture registers, and a control unit. The picture registers which are shown in Figure 8(b) comprise nine shift registers

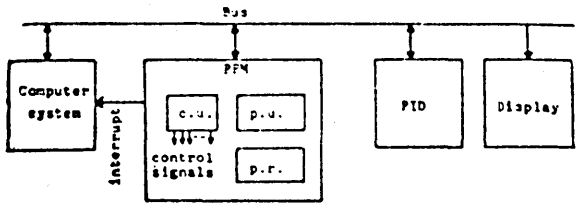


Figure 8(a)—Block diagram of parallel picture processing machine

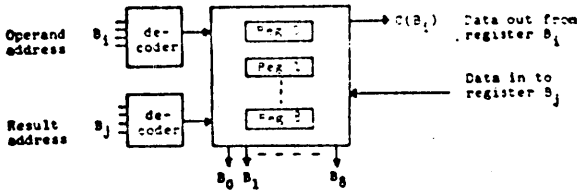


Figure 8(b)—Picture registers

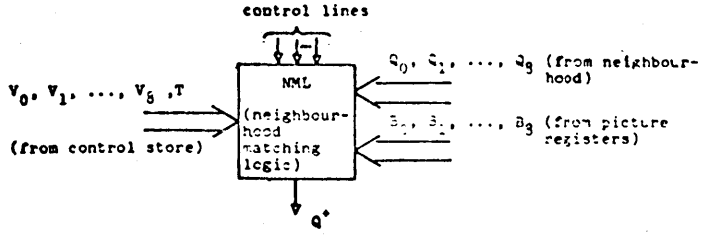


Figure 8(c)—Neighborhood matching logic

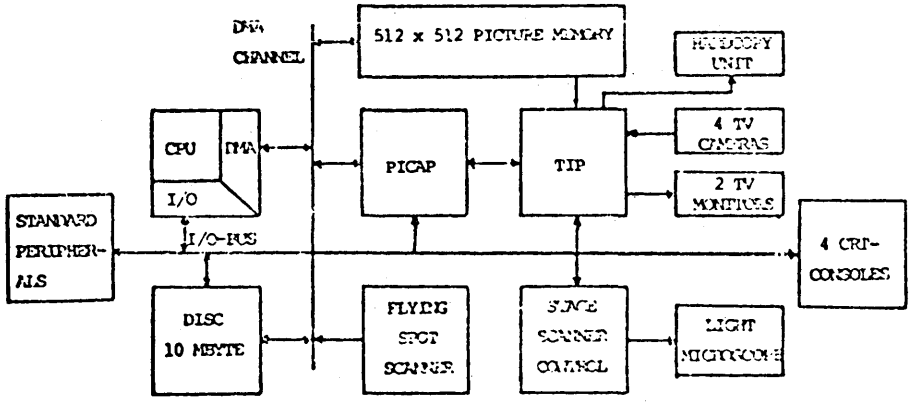


Figure 8(d)—Block diagram of PICAP computer system

capable of storing a picture. The main parts of the processing unit are the neighborhood matching logic (NML), shown in Figure 8(c), the line buffers (LB's) and neighborhood counting register (NCR) and coordinate register (COR). The demand for LB's stems from the fact that when a picture is stored in a picture register, only one picture point at a time is accessible. To be able to perform the local neighborhood operations, all nine neighborhood units must be accessed simultaneously. Therefore, two LB's are added. Note that the speed improvement is accomplished mainly at the expense of NML. The control unit is conventional in all respects except for the writable memory where the templates of an operation to be performed are stored. The parallel picture processor in Reference 9 has been linked with an ordinary minicomputer as part of the computation facilities of the PICAP picture processing laboratory[10] at the University of Linkoping, Sweden. The block diagram of the system is shown in Figure 8(d). The function of the minicomputer is twofold: (1) it controls the other devices and (2) it performs all nonpictorial data processing. PICAP which is a modified version of PPM is dedicated to the several tasks of the picture processing, such as fingerprint coding and malaria parasite detection. An important modification of PICAP has been the addition of a set of registers for the collection of measurements. The most important task of the picture processor is to produce measurements of application-dependent features. That is, to reduce the often enormous amount of information in an image to a set of feature measurements that can be handled by a conventional computer.

Each module in PPM is a synchronous sequential circuit with inputs from the 8 neighbors and the controller. All modules change states simultaneously.

STATE TRANSITION FUNCTION—Q(i,j)+=d(V,Q(i,j), Q(i,j+1), . . . , Q(i+1,j))
OUTPUT FUNCTION—U(i,j)=Q(i,j) [Moore model]

Since a Moore model is assumed, signal propagation through many layers of combinational logic is avoided and results in faster execution.

The system possesses gray scale capability since Q(i,j) can be many bits wide (say n) and can represent 2**n gray levels. This permits arithmetic operations on pictures. (Digital filtering, thresholding, image enhancement, etc.) In order to avoid long control words to fully specify the desired state transition function (extremely large number of state transition functions are possible, but few are useful in practice) a template matching scheme is proposed. Each module is capable of storing a list of templates, against which the current state vector is compared, and if a match is found a state change to the specified state occurs.

It is also possible to define arithmetic operations on pictures. For example:

$$f(P) = \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

can be used as the template to find the discrete Laplacian of a picture. Here Q(i, j)+=−4*Q(i, j)+Q(i−1, j)+Q(i, j−1)+Q(i+1, j)+Q(i, j+1). Operation on two or more pictures is possible by using multiple arrays with interarray communication. Conceptually this system can perform virtually any local operation. Unfortunately, hardware costs prohibit a fully parallel implementation of all features.

CDC flexible processor

Using the approach of distributed computation for designing a special purpose computer for image processing, the Control Data Corporation designed the Flexible Processor.[11] The Flexible Processor was developed for a large digital change detection system for concurrent processing of four channels of side-looking radar imagery. The Flexible Processor is a microprogrammable processor and uses random access memory up to 1024 words of 48 bits each for microprogram control. One special feature of the Flexible Processor is its data transmission which is shown in Figure 9. Four separate balanced party-line transmission channels are available. Each 32-bit channel can initiate and receive register-buffered transfers. A scanning system allows several Flexible Processors on a party-line to communicate with each other and with peripheral devices. For output transmission, a 32-bit output channel is provided which interfaces on balanced lines. A dual internal data bus system is used

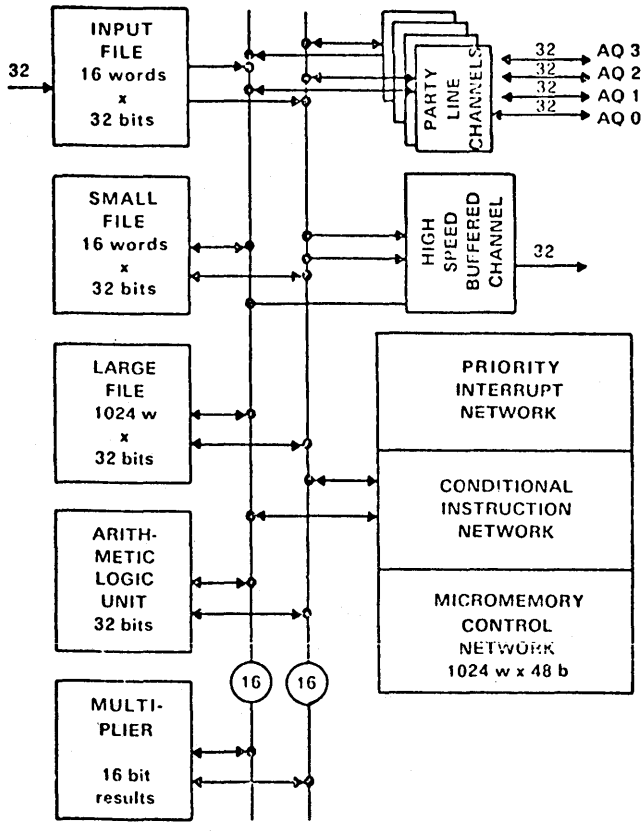


Figure 9—Block diagram of flexible processor by Control Data Corporation

to match data transfer speed to the speed of arithmetic logic. The arithmetic unit consists of an arithmetic logic unit, hardware network for conditional microinstruction execution, array hardware multiplier, specialized logic for square root and divide, and hardware priority interrupt mechanism.

TOSPICS

The TOSPICS, developed by Toshiba Company called Toshiba Image Processing System TOSPICS,[12,13] is an interactive image processing system which is a disk-based system and each operation is performed by the command input through a teletypewriter. The software system for the image processing system consists of the permanent and non-permanent resident system programs, the picture processors, the block common area, and a package of image processing programs. The system diagram of TOSPICS is shown in Figure 10. The special features of TOSPICS for image processing are that the image memory is commonly used in order to reduce the amount of data transmission and the parallel picture processor is constructed to perform certain

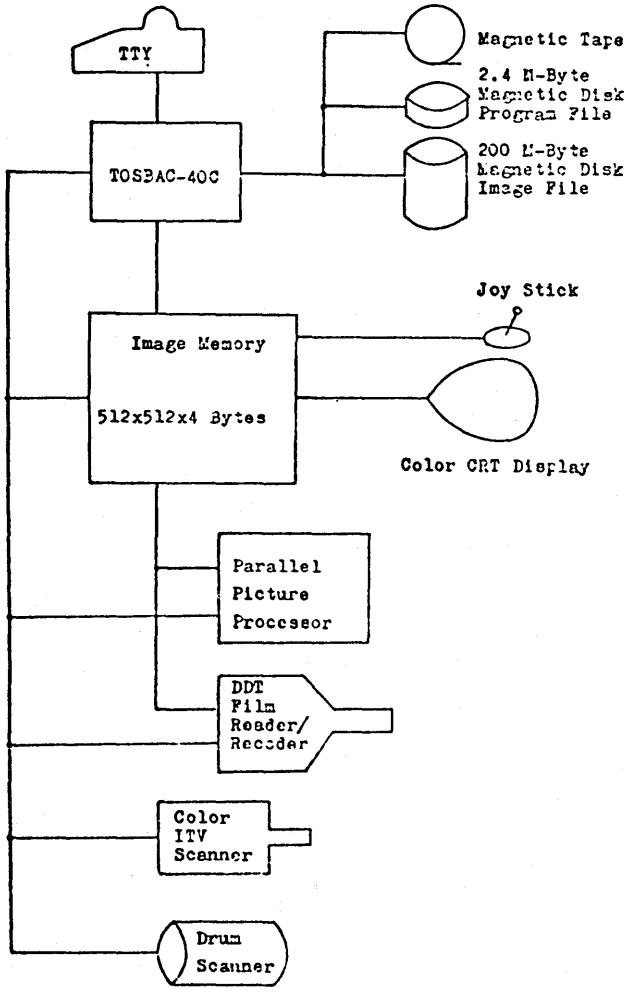


Figure 10—Block diagram of TOSPICS computer

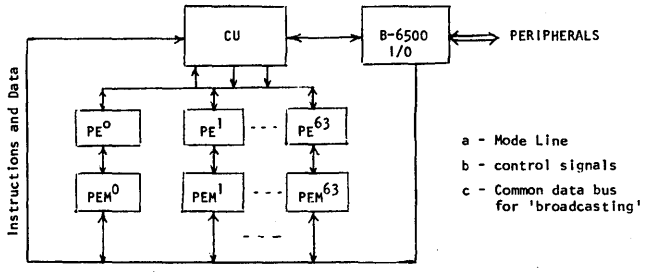


Figure 11—Functional block diagram of ILLIAC IV

programs at high speed. The parallel picture processor performs the program of spatial filtering at a speed of 1 pixel/1 μsec. The I/O devices of TOSPICS include a unique high precision flying spot scanner by a Double Deflection Tube (DDT) with 4096×4096 resolvable points and 32,000×32,000 addressable points.

Comparing the systems of Flexible Processors and TOSPICS, the basic ideas of parallel processing for image processing by the distributed computing approach are the same for these two systems. The difference between these two computer systems are the ways of utilizing processing elements. The TOSPICS uses the minicomputer, TOSBAC-40C, as an image processor and a parallel picture processor is attached to it. The parallel picture processor is used for certain programs, such as spatial filtering, affine transformations, histogram calculation, and density conversions.[12] The computer system of Flexible Processors uses the Flexible Processors as the processing elements. Each Flexible Processor is assigned for one task and several Flexible Processors are organized as the computer system.

ILLIAC IV

The ILLIAC IV[14,15] is a large array processor designed in the mid 60's to solve complex scientific problems involving arrays (matrices). The system was designed to have 256 processing elements (PE's) divided into four quadrants to achieve a total throughput rate of 1000 MIPS. However, due to cost overrun, only one quadrant has been built so far. The ILLIAC IV is patterned after the SOLOMON computer and employs one master control unit (CU) for instruction fetching, decoding and scalar operations. All processors in the array execute the same command (Figure 11).

Each processing element (PE) may be in one of the two modes: active/inactive, thus permitting some modules to be selectively disabled. Each PE has access to its own 2048 words of 64 bit memory called a PE memory (PEM). Each PE is a sophisticated arithmetic and logic unit capable of performing a wide range of operations. The PE's have been specially designed to perform floating point operations on 64 and 32 bit words very fast. The CU is also capable of executing its own instructions in parallel with the PE's. The CU has access to all the PEM's and fetches instructions from them. A broadcast facility permits common operands to be fetched by the C and 'broadcast' to all the PE's. Each PE has communication paths to nearest left and right neigh-

bors as well as to PE's at distance 8 on each side. The ILLIAC IV is under control of a Burroughs B-6500 processor which handles all I/O and operating system tasks in a conventional manner.

The ILLIAC IV is a powerful array processor for operations on matrices, vectors, etc. However, since most instructions on this system are geared toward 64 or 32 bit floating point operations while most image processing is done on integer arrays of 8 or 12 bit length, there is considerable mismatch and results in many problems. If only one pixel value is stored per ILLIAC word, most of the memory is wasted whereas packing results in execution time inefficiencies. (Eight-bit arithmetic has been recently incorporated in its hardware.[25])

The ILLIAC IV is organized as a linear array rather than as a two-dimensional array. The only routing signals being to the nearest neighbors much of the spatial connectivity relations have to be established by software.

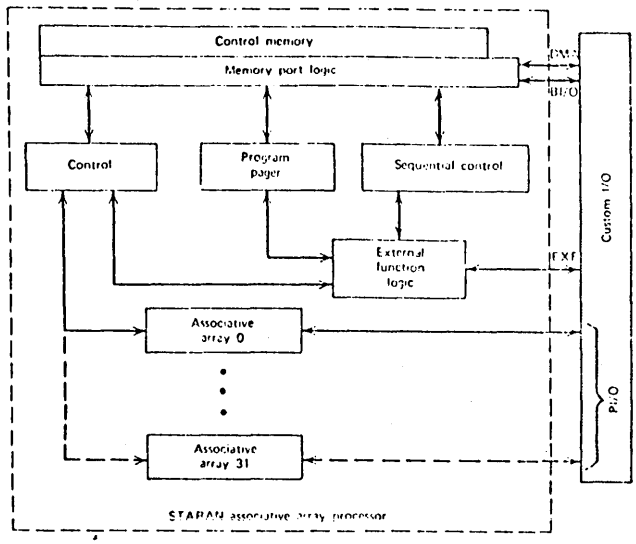


Figure 12—Block diagram of STARAN computer

STARAN

The STARAN[16] computer is a parallel processing computer built by Goodyear Aerospace Company in 1972. STARAN was not designed as a special purpose computer for image processing. But this computer was utilized for the task of resampling in the field of image processing in 1977.[17,18] The STARAN parallel processor is a *single instruction* stream *multiple data* stream (SIMD) processor.[19] The previously reviewed computers, such as, ILLIAC III, CIP 4, PICAP, and TOSPICS are also SIMD processors. The Flexible Processor is a SISD processor. The single most important element of STARAN is the associative array, which provides content addressability and parallel instruction execution capabilities. Most STARAN computing is done within a word of associative array memory. An associative array word is normally divided into fields of varying lengths by the programmer to suit the requirements of specific programs. The values of these fields then can be added, subtracted, multiplied, and divided within the word. The STARAN can perform the same operations as a sequential processor but with the added capability of performing these operations simultaneously on literally thousands of words in the associative processor arrays. A basic STARAN configuration contains an associative array. However, up to 32 associative arrays can be included in a single STARAN system. The block diagram of the STARAN computer is shown in Figure 12. The sequential controller provides off-line capabilities for assembling and debugging STARAN programs and control of STARAN error processing, diagnostic, and maintenance programs. Buffered I/O is available for tying different types of peripherals into the STARAN control memory. Also BI/O can be used to transfer blocks of data and/or programs between the STARAN control memory and host memory. The external function (EXF) logic facilitates coordination between the different elements of STARAN for special functions and simplifies housekeeping, maintenance, and test functions. By issuing external function codes to the EXF logic, elements of STARAN can control and interrogate the status of other elements. In general, the STARAN computer is powerful compared with the Flexible Processor and TOSPICS. However, only one kind of image processing task (resampling) has been carried out on this computer. Therefore, the effectiveness and efficiency of STARAN for all types of image processing and pattern recognition is still an open question.

Each STARAN array consists of 256 processing elements connected to a (256×256) bit multidimensional access memory. The array is controlled by a conventional sequential processor. The array memory and routing system permit memory access in many different modes. For example, it is possible to access in either column mode or row mode. Hence, we can process either all bits within a word in parallel or the same field of all desired words in parallel. Shifting capability has also been provided for various data manipulations such as the familiar butterfly computation for evaluating Fast Fourier transforms.

It is possible to use an associative processor as a parallel processor to operate on pixels simultaneously. For this the associative nature of the processor is not utilized but rather the parallel processing capability of the associative processor array. On the other hand it is possible to utilize the content addressibility of associative processors to simplify certain problems in image processing. Several illustrative examples can be found in Reference 22.

REMARKS

Consider the attributes of the bit-plane processing approach and the distributed computing approach. The bit-plane approach mostly uses Boolean operators as processors. Boolean operators work only on binary images which are not common in the real world. One way to get around this is to use several binary picture planes to represent the grey scale values of picture points. But, the complex software and additional memory requirement cause another problem and this problem limits the processing power of the

processor. The distributed computing approach appears to be more feasible considering the present state-of-the-art with respect to both software and hardware. Almost all the special computers discussed are only designed and tested in terms of signal-level processing[21] (such as image filtering, thresholding, thinning, and edge detection). There has been very little attention at the symbolic level, such as syntactic and semantic processing. A major drawback of previous computers is that the system's processors are not reconfigurable. The vast varieties of sensor types, applications, and image processing techniques, require that the image processing system (especially the parallel processor) be reconfigurable. Therefore, a generalized computer architecture which is reconfigurable under software control appears to be appropriate for the many applications of image processing. One of such a system has recently been proposed.[23] In the proposed computer architecture parallelism with the task is exploited by a multi-microprocessor parallel processor. In the meantime the operations of the sequential arithmetic processor are pipelined with the parallel processor under programmer control in certain tasks which can be decomposed into pipelined processing. Therefore, parallelism and pipelining are exploited at the same time in the proposed computer architecture.

In some special applications, the use of an auxiliary memory for image processing has been implemented in conventional computer facilities for image processing and pattern recognition research. One example is described in Reference 24. The computing facility is organized around a DEC PDP 11/45 digital computer with 32K core memory and 96K fast secondary memory, two disk drives, a magnetic tape drive, two cassette tape drives, a line printer, and a CRT monitor. The computer is not a special purpose computer, but the auxiliary memory is a special feature for image processing. The auxiliary memory was developed for image processing mainly because of the limited addressing range of the 16-bit minicomputer. Therefore, the limited addressable memory is difficult to use for implementing large programs. A memory controller has been built which can access up to 2^{32} bytes of memory and which now controls 64K 16-bit words of core memory. The controller is interfaced to the PDP 11/45 unibus through essentially three 16-bit registers. Two registers form an effective address of 32 bits, and the third register contains the data to be written into the memory or the data read from memory. Once the address is loaded on the two registers, data can be written into or read from memory in less than one microsecond. Since large arrays of data can be addressed without I/O to the disks, execution times improve. Also, computer programs become simpler since no sophisticated disk-core swapping software is needed. An experiment on rib identification in chest x-ray images was programmed once without using the auxiliary memory and once using this memory. On the average, the execution time improved a factor of five. Other software shows comparable execution time.

REFERENCES

1. Unger, S. H., "A Computer Oriented Toward Spatial Problems," *Proc. IRE,* October 1958, p. 1744.
2. McCormick, B. H., "The Illinois Pattern Recognition Computer-ILLIAC III," *IEEE Trans. on Electronic Computers,* EC-12, No. 5, 1963, pp. 791-813.
3. Stamopoulos, C. D., "Parallel Algorithms for Joining Two Points by a Straight-Line Segment," *IEEE Trans. on Computers,* June 1974, pp. 642-646.
4. Stamopoulos, C. D., "Parallel Image Processing," *IEEE Trans. on Computers,* Vol. C-24, No. 4, April 1975, pp. 424-433.
5. Duff, M. J. B., D. M. Watson, et al., "Cellular Logic Array for Image Processing (CLIP 2)," *P. R. 73,* Vol. 5, pp. 229-247.
6. Deutsch, E. S., "A Parallel Computer for Array Processing," *Info. Proc. 74,* pp. 94-97. This paper discusses CLIP 3 in detail.
7. "CLIP 3 Operating Manual," Image Processing Group, Dept. Phys. Astron., University College London, England, Rep. 73/4.
8. Duff, M. J. B., "CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor," Int. Joint Conference Pattern Recognition, Coronado, California, November 1976.
9. Kruse, B., "A Parallel Picture Processing Machine," *IEEE Trans. on Computers,* Vol. C-22, No. 12, December 1973, pp. 1075-1087.
10. Kruse, B., "The PICAP Picture Processing Laboratory," Third Int. Joint Conference on Pattern Recognition, Coronado, California, November 1976.
11. Allen, G. R., L. O. Bonrud, J. J. Cosgrove, and R. M. Stone, "The Design and Use of Special Purpose Processors for the Machine Processing of Remotely Sensed Data," Machine Processing of Remotely Sensed Data, LARS, Purdue University, 1973.
12. Shinada, H., H. Asada, T. Kondo, K. Mori, M. Kidode, and Numagami, "An Interactive Image Processing System—TOSPICS and its Application to Remote Sensing," *Proc. of Seventh Annual Symposium of Automatic Imagery Pattern Recognition,* College Park, Maryland, May 23-24, 1977, pp. 177-183.
13. Mori, K., H. Shinoda, and H. Asada, "Toshiba Pattern Information Cognition System—TOSPICS," *Toshiba Review,* No. 107, January 1977.
14. Barnes, G. H., et al., "The ILLIAC IV Computer," *IEEE Trans. on Computers,* Vol. C-17, August 1968.
15. Kuck, D. J., "ILLIAC IV Software and Application Programming," *IEEE Trans. on Computers,* Vol. C-17, August 1968.
16. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," *Proc. FJCC 72,* pp. 229-241.
17. Vocar, J. M. and R. O. Faiss, "Warp Processing Using STARAN," *Proc. of Workshop on Picture Data Description and Management,* Chicago, Illinois, April 21-22, 1977.
18. Rohrbacher, D. and J. L. Potter, "Image Processing with the STARAN Parallel Computer," *Computer,* August 1977, pp. 54-59.
19. Enslow, P. H., Jr., *Multiprocessors and Parallel Processing,* John Wiley & Sons, Inc., 1974.
20. Cornell, J. A., "PEPE—Parallel Processing of Ballistic Missile Defense Radar Data with PEPE," *Proc. COMPCON 1972,* pp. 69-72.
21. Cordello, L., M. J. B. Duff, and S. Levialdi, "Comparing Sequential and Parallel Processing of Pictures," Third Int. Joint Conference Pattern Recognition, Coronado, California, November 1976.
22. Ramesh, N. S. and K. S. Fu, "A Survey of Computer Architectures for Image Processing and Pattern Recognition," Tech. Rept. TR-EE 77-38, Purdue University, October 1977.
23. Keng, Janmin and K. S. Fu, "A Special Computer Architecture for Image Processing," *Proc. 1978 IEEE Computer Society Conference on Pattern Recognition and Image Processing,* May 31-June 2, Chicago, Illinois.
24. Akers, A. E., E. Persoon, and K. S. Fu, "A Virtual Memory Computer System for Image Processing," 1977 COMPSAC, November 8-11, Chicago, Illinois.
25. Hord, R. M. Private communication.

Experience with a picture processor in pattern recognition processing*

by BJÖRN KRUSE
Linköping University
Sweden

INTRODUCTION

In recent years much effort has been devoted to the study of picture processing algorithms and pictorial pattern recognition. The efforts have resulted in both theoretical understanding of and practical approaches to some of the basic problems involved.

Common to the areas of picture processing is the great need for processing power. One picture may contain as much as several million picture elements (pixels). A scene viewed through a common television camera for example, represents approximately two megabits of data renewed 25 times per second. To produce or analyze data at such rates is completely beyond the capacity of the general purpose computer. Even if the speed is increased by a factor of ten, which may be anticipated in the future, the GPC will fall short when faced with real-time applications. The problems stem from the fact that pictures represent a new type of data, orders of magnitude richer in information than that for which the GPCs architecture was designed.

Many attempts have been made to design a computer architecture that is more suited for picture processing[1–7] and in some cases the machines have also been constructed.

The design of the PICAP processor has evolved through a number of generations. Starting with a simulation study in 1971,[8] the first version of the processor was built in 1973.[4] Gradually as experience with the processor accumulated new features were incorporated such as automatic histogramming and collection of various measurements made on the picture. The final version of PICAP which will be described here has been operational since the beginning of 1975.

PICAP AND ITS ENVIRONMENT

The following paragraphs give a brief presentation of the PICAP Picture Processing Laboratory as a background to the description of PICAP's architecture. For a complete description of the laboratory see Reference 9.

PICAP operates in close connection with a conventional minicomputer of quite standard configuration. Standard peripherals, such as alphanumeric and graphical displays, and a multi-user disc operating system are the main features. PICAP and the computer are also connected to an advanced television input/output interface, TIP[15] (see Figure 1). Both PICAP and TIP are controlled by the computer which besides its controlling function performs all computations of non-pictorial nature.

Through TIP PICAP has access to an entire TV field of 512×640 pixels from which a randomly positioned window of 64×64 4-bit pixels may be extracted at full video rate. Both the sampling density and gray-level range in the digitization process are controlled by a set of programmable parameters. Thus the window may in one frame represent the entire field coarsely sampled with the full dynamic video range mapped onto the 16 level gray-scale and in another frame it may represent a small subpicture densly sampled in a certain slice of the video range. Consequently a scene may be analyzed on many levels starting with a coarse overview and ending with a detailed analysis in full resolution. These features correspond in a way to the vision systems found in nature where only a limited retinal area, the fovea, have high spatial resolution and where a scene is sequentially scanned by mechanical eye movements.

There are four video channels available in TIP of which two are used for the present. One camera is permanently mounted on a light microscope which is supplied with a computer controlled stage scanner and the other is mounted on an optical bench so that ordinary slides may be read.

THE ARCHITECTURE OF PICAP

PICAP belongs to the category of "single-instruction-stream multiple-data-stream" (SIMD) processors. Unlike many processors used for pattern recognition, it is fully programmable with an instruction set especially tailored for efficient processing of both gray-level and binary pictures. Equally important as the feature of programmable picture into picture transformations is PICAP's capability to extract measurements. In all pattern recognition tasks the (often very redundant) input data has to be reduced into a set of adequate descriptors that describe its relevant features, such as gray-scale distribution, object position and shape. It is very important that this can be done efficiently since these

* This work is supported by the Swedish National Board for Technical Development (STU).

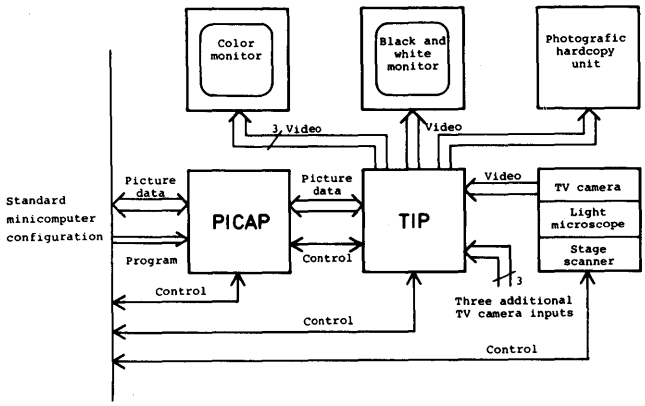


Figure 1—Processor environment

descriptors, measurements, form the basis for a subsequent classification. From the processor's point of view the measurements constitute the final result. The picture transformations only serve to enhance the features that finally are to be measured.

The 64×64 4-bit window mentioned in the preceding section constitutes the basic data unit that PICAP operates on much in the same way as the 16-bit word is the data unit for common minicomputers. Figure 2 shows schematically the two main types of processor action on the picture data, *multiple* picture processing and *neighborhood* processing. In the multiple processing mode two, X and Y, or more (up to nine) pictures are operands giving a result Z, the pixels of which are determined by the corresponding pixels in X and Y and the actual operation. Thus

$$Z_{ij}=F_C(x_{ij}, Y_{ij}, \ldots)\ i,j=1, 2, \ldots, 64$$

The mapping function F_C is defined by C, a set of parameters which is supplied by the instruction. Typical multiple mode instructions are weighted summation (e.g., subtraction) of pictures and masking.

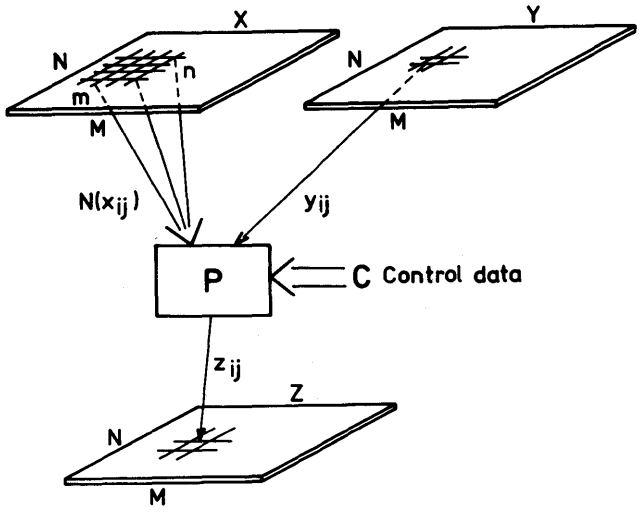


Figure 2—Schematic operation of the processor

The neighborhood operation mode applies to a single picture, X, at a time. In this case the output pixels of array Z are determined by corresponding pixels in X and their immediate surrounds, the 3×3 neighborhoods $N(x_{ij})$.

$$Z_{ij}=F_C(N(x_{ij}))\ i,j=1, 2, \ldots, 64$$

The actual neighborhood operation is determined by the parameter set C and it is performed identically on all neighborhoods of X. Convolutions, shrinking and labelling of pictures are typical examples.

Note that the actual mapping, symbolized by F_C, is identical in the two modes. It is only the operands that differ.

In addition to the instructions belonging to the categories above there are trivial instructions, such as input/output and shift of pictures, that will not be discussed here.

The main units of the processor are shown in Figure 3. The picture memory PM is a set of nine separate picture registers $R_0, R_1, \ldots, R_8$ into which any window of the input TV field may be loaded. The registers are also used for intermediate results much in the same way as the registers of common CPUs are. The MLA unit in which the actual processing is performed takes its operands either from the nine picture registers (multiple mode) or from the neighborhood registers $Q_0, Q_1, \ldots, Q_8$ of the LB unit (neighborhood mode). The parameter that determines the actual processing resides in CTM which essentially is an extension of the instruction register not shown in the figure.

Simultaneously to the processing of pictures the MLA unit collects instruction dependent measurements that are stored in a set of thirty-three registers, MR. To enable conditional branches in the instruction flow to the processor the contents of these registers are made available to the controlling computer at the end of each executed instruction. In a way the measurement registers are the analogue of the status register in conventional computers in that they describe the outcome of an instruction.

As has been stated above PICAP is a SIMD processor since a single instruction determines the operation performed on a large set of operands, the pixels. Physically, SIMD processors may be designed with different degrees of hardware parallelism and this is especially apparent for picture processors where there are three distinct levels on which the parallelism may be introduced, the pixel, the neighborhood and the picture levels. On the pixel level there is a choice on the number of bits that can be processed in parallel. In PICAP this is four, contrary to most existing picture processors that are binary. On the neighborhood level the degree of parallelism is equal to the number of neighborhood pixels that may be accessed and processed simultaneously which in most existing processors including PICAP is nine, a 3×3 neighborhood. Finally on the picture level the choice is on the number of neighborhood processing units. In PICAP there is one, the MLA unit, which means that the neighborhoods are processed sequentially. The level on which the great hardware effort should be made is a controversial question which will be commented on in the last section.

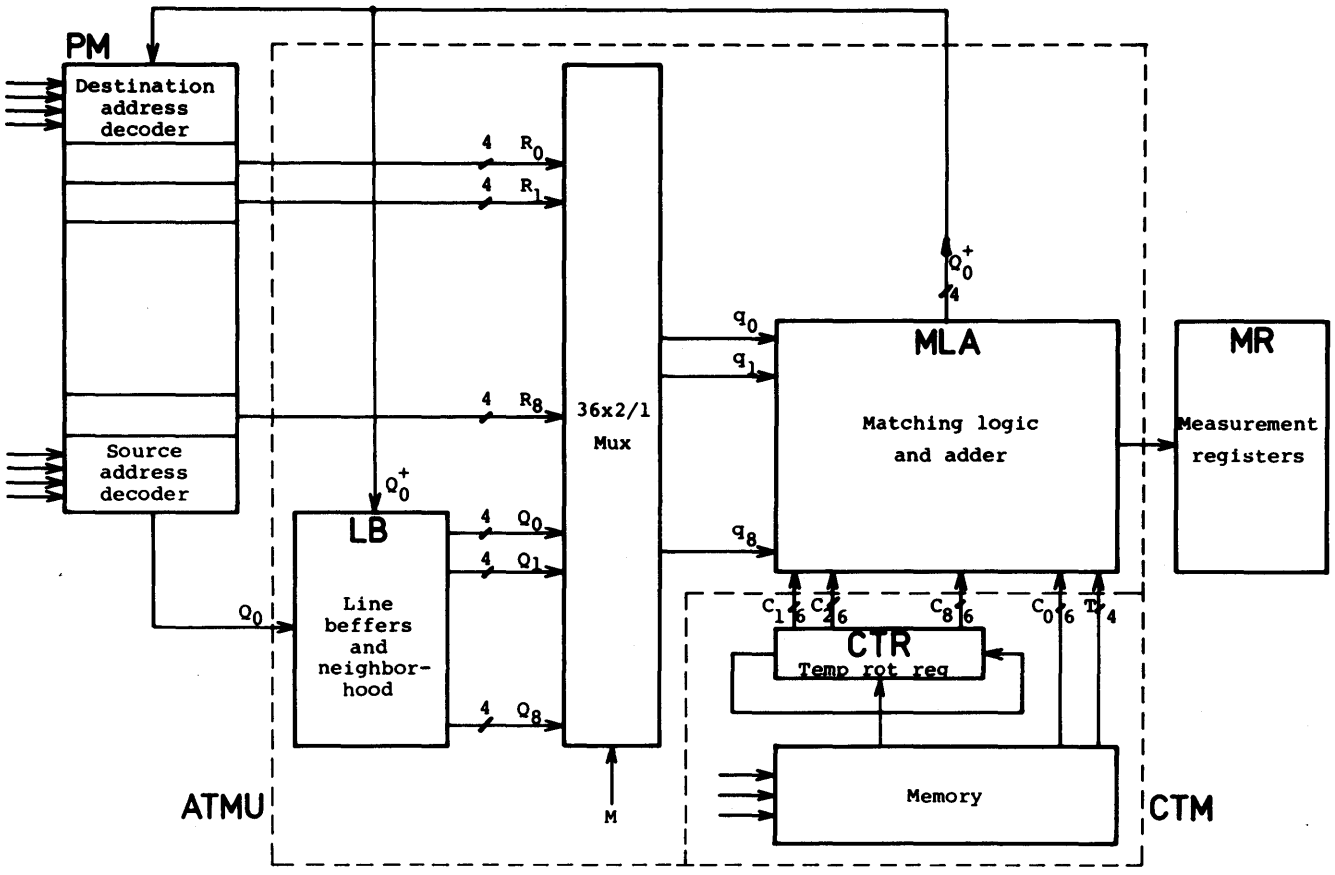


Figure 3—Matching logic, adder and line buffers

THE INSTRUCTION SET

There are two main PICAP instruction categories: the logical and the arithmetical instructions. The instructions are described in detail in References 9 and 10 and only a summary will be given here. Their characteristics will be described as applied in the neighborhood operation mode. The implications for the multiple mode operation are obvious.

The logical neighborhood instructions are based on generalized 3×3 templates called *condition templates* consisting of nine subconditions one for each of the neighborhood elements. Each instruction may contain up to eight such condition templates. In the course of execution the picture neighborhoods are all subjected to a test whether they do or do not belong to the class of neighborhoods defined by the set of condition templates of the instruction. If a neighborhood matches a certain template its center pixel is given a new value, T, that is supplied by the instruction. *All other pixels remain unaffected.* In case of multiple hits a simple precedence rule determines the new value. The entire transformation may be thought of as an associative multiple search in two dimensions for pixels whose neighborhoods correspond to certain features as defined by the templates.

The subconditions C_K are composed of a relation R_K and a number n_K, $C_K=\langle R_K, n_K\rangle$. The condition is fulfilled iff the pixel q_K is related to n_K by R_K. Four different relations may be used: equal to (E), less than (L), greater than (G) and the universal relation (X=don't-care). The subconditions are defined over the 3×3 neighborhood forming a condition template

$$C \equiv \begin{matrix} C_4 & C_3 & C_2 \\ C_5 & C_0 & C_1 \\ C_6 & C_7 & C_8 \end{matrix} \quad T$$

A certain neighborhood matches a condition template iff all its element fulfill their corresponding subconditions. To each condition template there is associated a transition value T, the value into which the center pixel is transformed in case of a match.

The format of the instruction is shown in Figure 4. It is a variable length format as the number of condition template is not fixed. The instruction-head contains a modifier field and the addresses of the source and destination registers besides the operation code. A number of variations to the instruction's basic properties as described above may be specified in the modifier field (see Figure 5). For example,

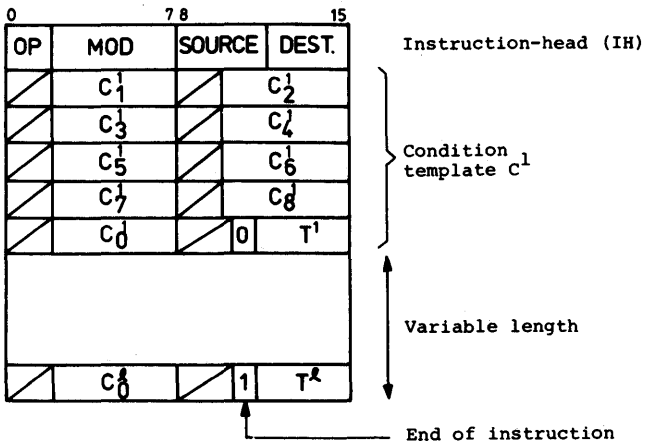


Figure 4—Format of logical instructions

the modifier M determines the mode of operation, multiple or neighborhood mode. The other modifiers apply to isotropic application of the templates ($R4$ and $R8$), loop operations (It), extraction of the first matching neighborhood (In) and sequential picture operations (S). A full descriptions of these are given in Reference 10 which also contains a number of typical operations. A trivial example is given below.

Example 1: Neighborhood dependent thresholding.

X	Gn	X			X	X	X	
Gn	Gn	Gn	1		X	X	X	0
X	Gn	X			X	X	X	

The first template of the instruction covers the class of five element neighborhoods whose elements have a value greater than n (Gn). Due to the precedence rule, center points that belong to such neighborhoods will aquire the value 1 irrespective of the second template. As the latter covers all possible neighborhoods the remaining pixels will get the value 0. (End of example.)

Compared to the logical instructions the arithmetical ones are conceptually much simpler. They have the form of a convolution between an array of coefficients defined over a 3×3 neighborhood and the picture array. The computations are followed by normalization with a factor 2^{-n} so that the result will not overflow. Hence the center pixel of the neighborhood is replaced by the value of the expression

$$\left(\sum_{i=0}^{8} m_i q_i\right) 2^{-n}$$

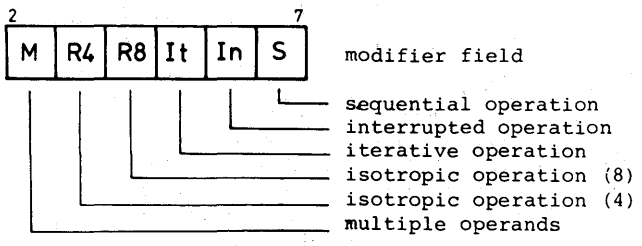


Figure 5—The modifier field

The coefficients $\{m_i\}$ and the normalizing exponent n have eight and four bits precision respectively. The instruction format is similar to the logical instruction format.

The following example shows a simple contrast enhancement operator.

Example 2: Contrast enhancement.

Coefficient array:			Normalizing factor:
−1	−1	−1	
−1	40	−1	2^{-5} (1/32)
−1	−1	−1	

The operator has unit again for low spatial frequencies and enhances high frequencies.

MEASUREMENT EXTRACTION

The picture transformations described in the preceding section are accompanied by a fairly large set of automatically performed measurements.

- Neighborhood counting
- X-Y extent determination
- Gray-scale maximum, minimum and average
- Histogram collection

Neighborhood counting

To each condition template of a logical instruction is associated a counter which is incremented for each occurrence of a matching neighborhood. Hence locally countable properties such as Euler number, area and perimeter are easily extracted.

X-Y extent determination

To provide for extraction of positional information there is a set of four registers $X_{\min}$, $X_{\max}$, $Y_{\min}$ and $Y_{\max}$ that gives the maximum and minimum coordinates on both axes for the occurrence of a matching neighborhood. The set of coordinates defines the minimum rectangle aligned with the axes that enclose a set of features which is described by the templates.

Gray-scale maximum, minimum and average and histogram collection

The gray-level distribution and its maximum, minimum and average are collected in a set of 19 registers HC_0, HCl, . . . , HC_{15}, $G_{\max}$, $G_{\min}$ and G_{av}. Any transformation of a picture results in an updated set of gray-scale measurement.

It should be noted that the inclusion of automatic measurements is especially simple in a processor where the pic-

ture is sequentially processed since it amounts to little more than a straightforward counting procedure.

EVALUATION AND PERFORMANCE

During the years that PICAP has been operational it has been employed in many different pattern recognition projects. A subset of these form the basis for the following evaluation of the instruction set. The subset is chosen to contain quite dissimilar applications in order to reflect the operational use of PICAP in different contexts. The evaluation is based on three programs: printed circuit board inspection, malaria parasite detection and fingerprint coding. The object of the statistical analysis below has been to quantify the use of the instruction categories, modifiers etc. and most of all to see how often the controlling minicomputer has been used for picture processing.

The printed circuit inspection program tests the circuits with respect to the minimum tolerable conductor widths and separation. It incorporates necessary adaptive thresholding of the gray-level input pictures[11] and generation of a pseudo color output picture that indicates the defects. The nominal width of the conductors and insulators is checked by an appropriate number of erosion and swelling operations. The statistics are shown in Figure 6.

The statistic material shown in Figure 7 is the pooled statistics from two separate programs for classification of and core location in fingerprints. For a complete description of these programs see References 12 and 13. The fingerprints are read by a TV camera. Through the use of appropriate direction sensitive filter operators the ridge and valley pattern is extracted. Then syntactical methods are used to compute the category assignment and the location of the fingerprint core.

The malaria parasite detection program[14] is the most advanced of the application programs. Biological specimens of malaria-contaminated blood cells in a thin smear on a microscope slide are analyzed. During the scanning procedure the infected blood cells are located and the parasites are classified according to their stage of evolution. One test evolves the analysis of 5000-6000 red blood cells which takes approximately five minutes for a skilled microscopist whereas the automatic program takes eight minutes. Figure 8 shows the statistics. The descriptions above are of course very schematic and do not at all give full credit to the actual sophistication of the programs.

It is apparent from the pooled statistics in Figure 9 that the predominant instruction category is the logical instruction. This category accounts for over 70 percent of all executed instructions. It is less dominant in the case of the malaria parasite program (55 percent) presumably because the source data is inherently non-binary. The picture transfer and arithmetical instruction types are equally frequent, accounting for approximately 15 percent each, whereas the shift instruction is virtually never used. As a shole the malaria parasite program exhibits greater balance between instruction types, reflecting the general type of processing required.

The modifiers of the logical instructions are used in approximately 50 percent of the cases which means that the nonmodified, i.e., the one-operand local, logical instruction accounts for 50 percent. Among the modifiers the isotropic ($R8$), and the multiple operand modifiers are used most frequently. Of the eight available templates in the instruction format only one or two are needed in nearly 90 percent of the cases. The one- and two-template instructions account for 44 percent each.

The picture transfer instructions, of which the input type is predominant, are second-most-frequently used. It is an interesting fact that of all executed instructions input from the TV field to PICAP accounts for over 10 percent which means that on the average every tenth instruction inputs a new picture. This emphasizes the need for fast transducers. Loading of a picture should not be slower than the execution of one instruction if the processor is to be kept busy 90 percent of the time which seems to be a reasonable figure. Another interesting fact is that transfers of pictures to the host computer's memory are virtually nonexistent (less than 1 percent) which means that essentially *all picture processing operations are performed in PICAP*. As PICAP programmers always have the option of processing the pictures in the host computer this indicates that the chosen instruction and measurement sets are very well adapted to the task set.

The arithmetical instructions are used to approximately the same extent as the transfer instructions, 14 percent. Multiple operand instructions such as addition/subtraction of pictures are relatively infrequent, 12 percent.

Among the picture registers R_0 is most frequently used. In 68 percent of the instructions R_0 appears as either source or destination register. R_0 and R_1 together account for over 80 percent of the cases which indicates that the number of nine registers is more than sufficient.

To measure the performance in terms of the execution time for the basic instructions is always unsatisfactory since it obviously depends on many other factors such as the speed and versatility of the picture transducer, the host computer and its operating system. The best figure of merit for the PICAP system is given by the malaria parasite program which, as has been stated above, has a performance of a skilled microscopist. The processing speed including input, feature extraction and classification of the parasites is 10-15 blood cells per second.

Finally, for the sake of completeness the execution time for the basic one-template and two-template logical instructions are 1.2 and 1.5 μs/pixel respectively. The execution time for arithmetical instructions are 1.2 and 3.6 μs/pixel for a smoothing and an arbitrary convolution respectively.

CONCLUSIONS

The PICAP architecture and instruction set has been shown to cover the needs for solving typical pictorial pattern rec-

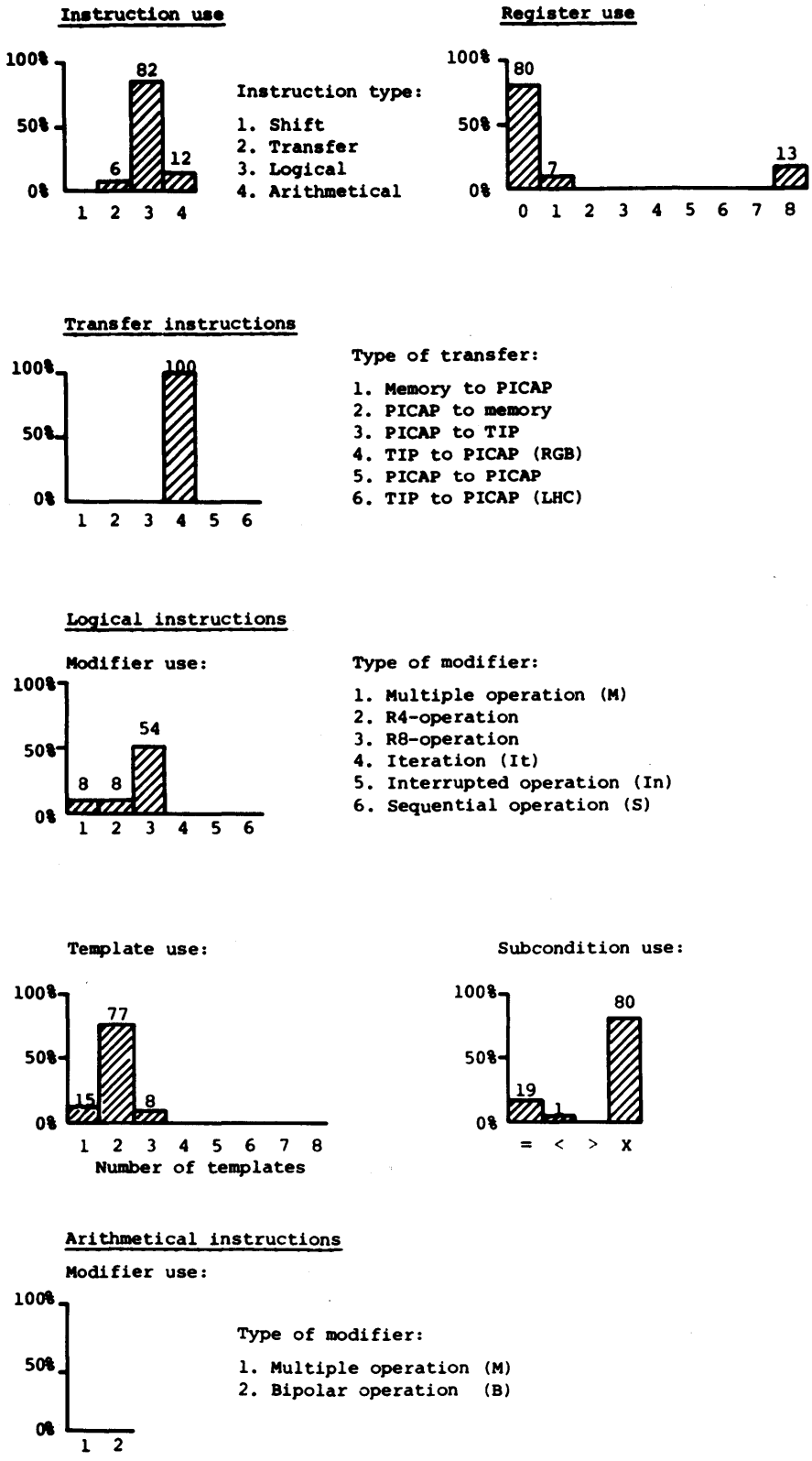


Figure 6—Statistics of the PC-board inspection program

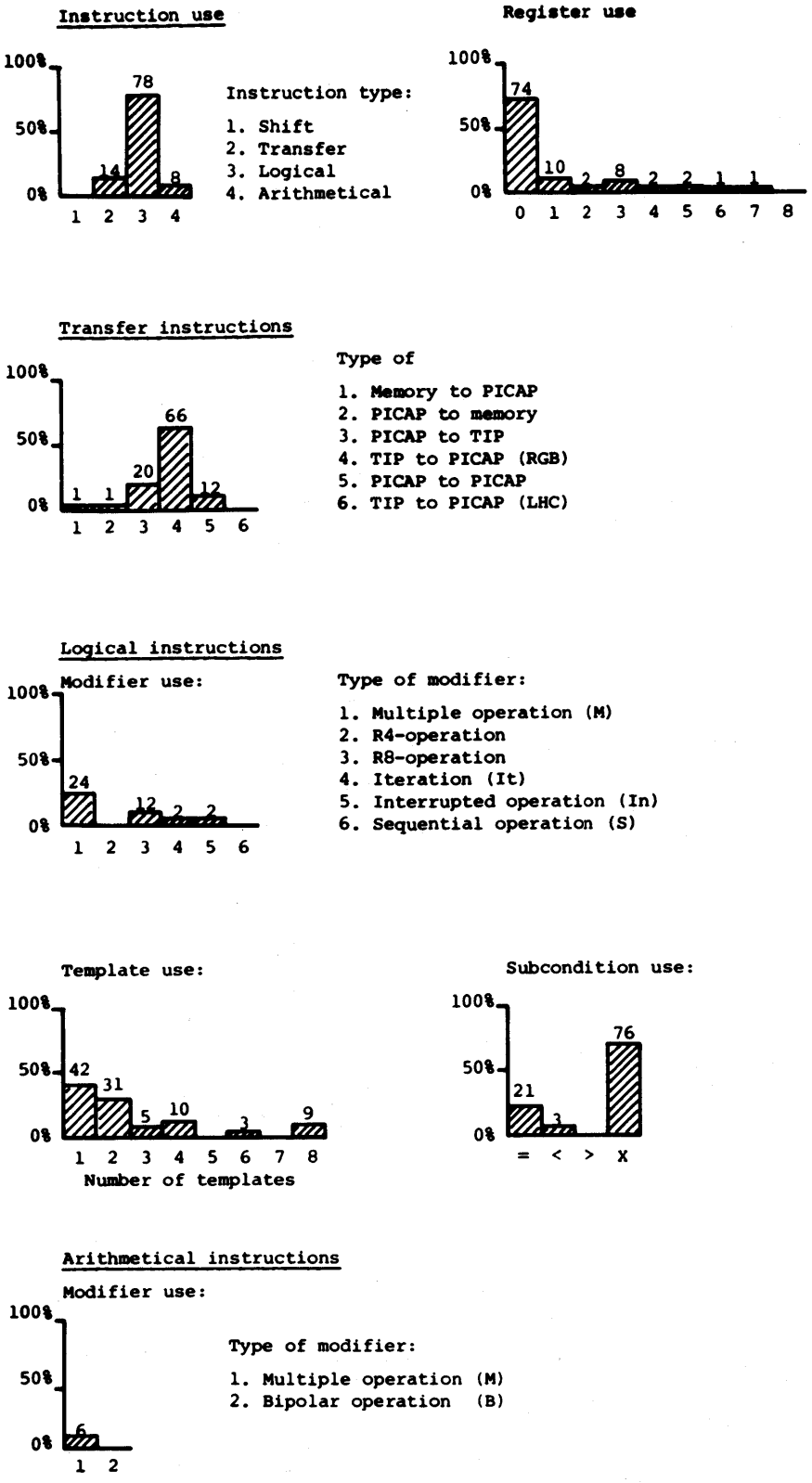


Figure 7—Statistics of the combined fingerprint classification

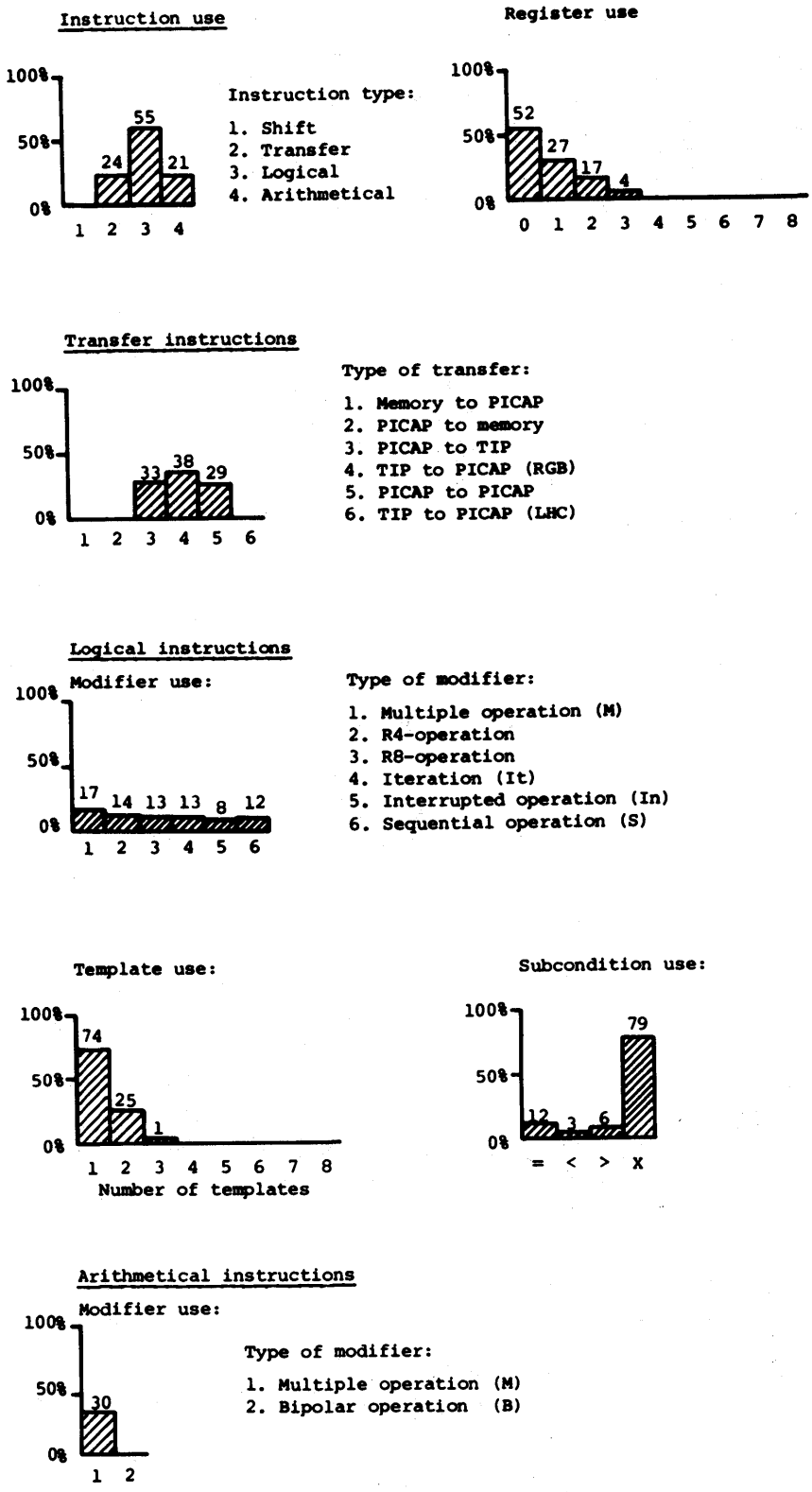


Figure 8—Statistics of the malaria parasite detection program

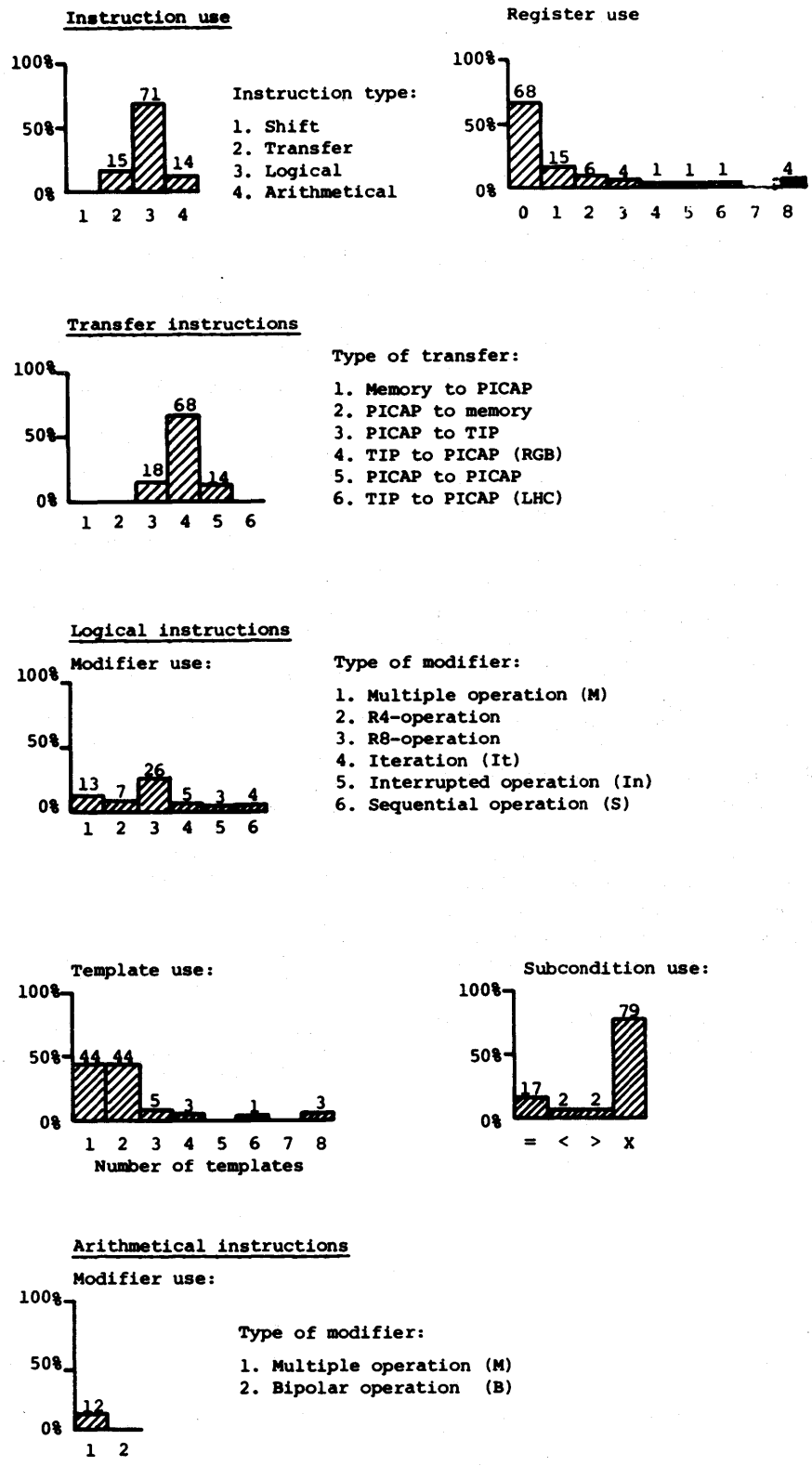


Figure 9—Pooled statistics

ognition applications very well. The performance in an advanced task like malaria parasite detection, equals that of a skilled microscopist.

The fact that on the average only nine operations are performed on each input picture is not explained by the well-adapted instruction set alone. It is also a consequence of the frequent use of several levels of sampling density. This means that featureless parts of the input scene often may be located in a coarse survey at an early stage of the recognition procedure. Since such parts need not be further processed the average number of operations per picture is lowered.

The high frequency of input instructions stresses the need for fast picture transducers. On the other hand, the speed requirements of a processor with PICAP's characteristics is limited to not much more than nine times the speed of the system's transducer.

The ability for fast collection of various measurements is very important. The ease with which such facilities are incorporated in PICAP, using not much more than a simple counter for each measurement, speaks strongly in favor of physically sequential machines.

In processors where parallelism on the picture level is used, corresponding measurements have to be extracted by a cumbersome programmed procedure. The number of program steps for such an extraction may be several orders of magnitude larger than the initial number of processing steps that produced the feature to be extracted. Therefore the high inherent speed of truely parallel machines in picture-picture transformations is very much impaired in pattern recognition applications.

In fact, PICAP with its low order of complexity can in several cases compete with more elaborate parallel designs. Another reason for this is that the sole neighborhood processing unit of a sequential processor may easily be made more powerful than the corresponding multiple units in a parallel design.

Finally, the continuing advance of IC manufacturing makes the future look bright for practical, high-performance picture processors.

ACKNOWLEDGMENTS

The author wishes to thank Dr. K. Rao, K. Balck and P. Rowa for the application programs and Prof. P. E. Danielsson for suggestions and support during this work.

REFERENCES

1. Unger, S. H., "A Computer Oriented Towards Spatial Problems," *Proc. IRE*, Vol. 46, 1958, pp. 1744-1750.
2. McCormick, B. H., "The Illinois Pattern Recognition Computer," *IEEE Trans. Electron. Comput.*, Vol. EC-12, Dec. 1963, pp. 791-813.
3. Golay, M. J. E., "Hexagonal Parallel Pattern Transformations," *IEEE Trans. Comput.*, Vol. C-18, Aug. 1969, pp. 733-740.
4. Kruse, B., "A Parallel Picture Processing Machine," *IEEE Trans. Comput.*, Vol. C-22, No. 12, Dec. 1973, pp. 1075-1087.
5. Gray, B. S., "The Binary Image Processor and its Applications," Information International Inc., Los Angeles, Calif. 90064, 90365-5C-Jan. 1972.
6. Duff, M. J. B., D. M. Watson and E. S. Deutsch, "A Parallel Computer for Array Processing," *Image Processing 74*, Proc. IFIP Cong. 74, 94, 1974.
7. Duff, M. J. B., "Clip 4: a Large Scale Integrated Circuit Array Parallel Processor," *Third Intern. Joint Conf. Pattern Recognition* 1976, pp. 728-733.
8. Kruse, B., (in Swedish) "Simulering och projektering av en ytmaskin," Swedish Research Institute of National Defense FOA 3, rapport C3716-E5, June 1972.
9. Kruse, B., "The PICAP Picture Processing Laboratory," *Third Intern. Joint Conf. Pattern Recognition* 1976, pp. 875–881.
10. Kruse, B., "Design and Implementation of a Picture Processor," Linköping Studies in Science and Technology Dissertation, No. 13, April 1977.
11. Weszka, J. S., R. N. Nagel and A. Rosenfeld, "A Threshold Selection Technique," *IEEE Trans. Comput.*, Vol. C-23, 1974, pp. 1322-1326.
12. Rao, C. V. Kameswara and K. Balck, "Type Classification of Fingerprints: A Syntactic Approach," LiTH-ISY-I-077, Dept. Electr. Eng., Linköping University, Sweden, 1976.
13. Rao, C. V. Kameswara and K. Balck, "Finding the Core Point in a Fingerprint," LiTH-ISY-I-0079, Dept. Electr. Eng., Linköping University, Sweden, 1976.
14. Rowa, P. and D. Antonsson, "Automated Malaria Parasite Detection," LiTH-ISY-I-0185, Dept. Electr. Eng., Linköping University, Sweden, 1977.
15. Cronström, S. "A Description of a TV Interface for a Picture Processing Machine and an Introduction to Television," LiTH-ISY-I-0083, Dept. Electr. Eng., Linköping University, Sweden, 1977.

Design of local parallel pattern processor for image processing

by KEN-ICHI MORI, MASATSUGU KIDODE, HIDENORI SHINODA, and HARUO ASADA

Toshiba Research and Development Center
Kawasaki, Japan

INTRODUCTION

Several image processing machines have been designed and some of them already implemented. The Illiac III by McCormick,[1] hexagonal processor GLOPR by Golay,[2] and binary image processor BIP by Gray[3] are examples. The idea of parallel processing hardware has proved its feasibility, appealing to a number of industrial applications. Special purpose image processors, such as the optical character reader,[4] the white blood cell differential count analyzer,[5] the remotely sensed data analyzer[6,7,19] and the computer tomography machine[8] have their characteristic architectures according to their purpose. General purpose image processors have been proposed by several research groups[9–11,18] and are demonstrating their ability.

The problem now to be solved is determining what architecture of image processing machines is superior in the cost performance ratio. Trade offs in various aspects like speed, generality and economy, should be discussed. Our approach is a microprogrammable local parallel pattern processor (PPP) which has several basic image processing functions in a simple hardware. In the second section, major concepts considered for the design of the PPP will be discussed, and in the third section, the details of the implementation of the basic image processing functions are described. Some simple examples of programming the PPP will be shown in the fourth section, but the details of command formats, application softwares for remotely sensed data analysis, medical image processing, etc. can be found in other papers.[12–14]

DESIGN CONCEPTS

The PPP has several features in design concepts for improving the image processing cost performance ratio. These features contribute to speeding up major calculations in image processing to approximately 100 times faster than by the conventional digital computers, and to reducing the hardware cost and complexity.

Image memory

The PPP is directly connected to the image memory unit of the interactive image processing system TOSPICS (**TO**Shiba Pattern Information Cognitive System) shown in Figure 1. In the TOSPICS, the image memory unit plays five roles, as high speed buffer memory for image input/output devices, refresh memory for color display monitors, working memory for the PPP, buffer memory for a magnetic disk unit and external bulk memory for the host mini-computer. This configuration minimizes the "idle" data transfers among hierarchical levels of the memory. The time for those data transfers generally share a large fraction of the total throughput time by conventional computer configurations, which are not necessary for the essential calculations in the image processing.

Basic image processing functions

There are several types of computation that are frequently used in image processing. Most of such operations take a large amount of time. The parallel image processing machine should be fast enough to execute those basic operations for improvement in the cost performance ratio.

Those basic image processing functions are listed as follows:[15]

(1) Two Dimensional Convolution

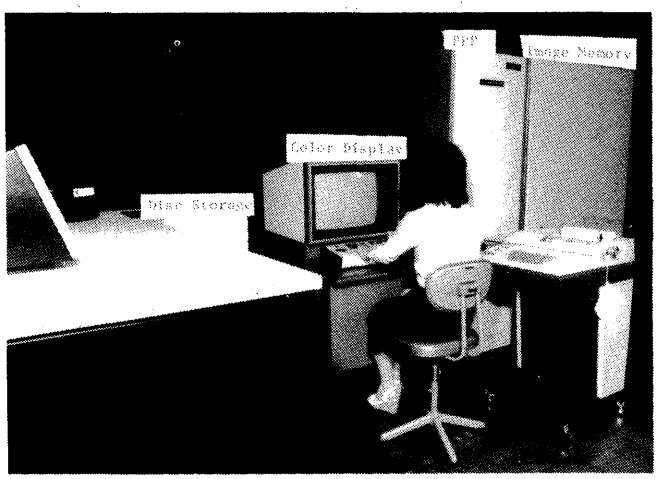
$$G(X,Y)=\sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}F_{ij}(X,Y) \tag{1}$$

where $F_{ij}(X,Y)$ is a neighboring pixel of input image data around the coordinate value (X,Y), and W_{ij} is an element of weighting matrix of size $m\times n$. This computation is used in such image processing operations as: convolution, spatial filtering, general linear algebra computations, correlation, template matching, texture analysis, K-L transformation of image,[13] and scene classification by matched filter or maximum likelihood method. Many operations used in spatial frequency domain analysis, such as restoration, enhancement, reconstruction from projections can be similarly expressed by Eq. (1).[16]

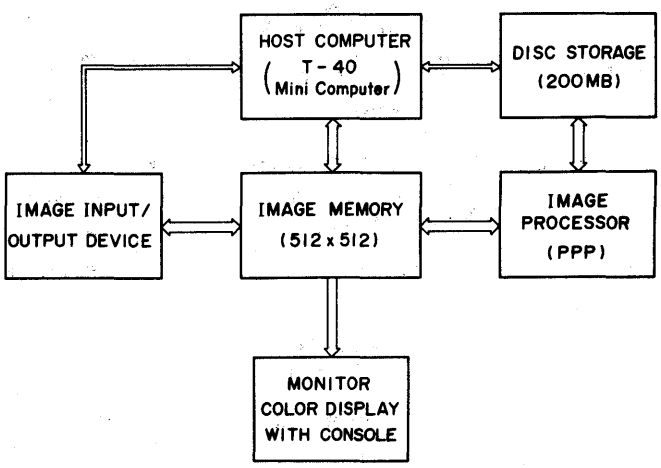
(2) Point Mapping

$$G(X,Y)=\phi\{F(X,Y)\} \tag{2}$$

where ϕ is a single-valued (usually nonlinear) function. Point mappings are used for such operations as: enhancement,



(a) TOSPICS Overall View



(b) TOSPICS Blockdiagram

Figure 1—Overview and blockdiagram of TOSPICS

nonlinear filtering, gray scale mappings, multilevel thresholding, pseudo-color generation, correction of intensity nonlinearity of image sensors and displays.

(3) Linear Coordinate Transformation

$$\begin{cases} G(X,Y)=AF(X,Y) \\ A:\begin{pmatrix} X' \\ Y' \end{pmatrix}=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix}+\begin{pmatrix} e \\ f \end{pmatrix} \end{cases} \quad (3)$$

where A is affine transformation specified by a linear coordinate transformation matrix and shift values. This operation is used for rotation, size change and registration of image.

In addition, logical filtering, region labeling, histogram generation and pixel operations are frequently used. These basic image processing functions are greatly speeded up by the use of specially designed circuits.

Local parallel vs. fully-parallel

There are two types of parallelism in image processing. The most popular idea is a two dimensional array of identical pixel processing elements, which works simultaneously on independent data stream per processing element. However, the image processors on the fully-parallel basis using a large number of LSI calculation elements[17] have restricted data communication between only adjacent neighboring pixels. This architecture will limit the applicability of the image processing operation of Eqs. (1) and (3), or will need excessive time to access neighboring image data, which cancels the speed-up effects brought about by the fully-parallel basis.

A pixel generally has stronger dependency to neighboring pixels than to those in the distance. The range of neighboring pixels depends on the nature of operations, but it falls between 3×3 and 32×32 pixels surrounding the center pixel. In the PPP, the local parallel processing, i.e., parallel accessing and calculation within local dependency up to $n\times n$ neighboring image data, is carried out in the specially designed circuits. Then, the center position of local parallel operation is shifted sequentially to the next position. The combination of local parallel processing and the sequential processing fits the serial data transfer characteristics of image data input/output devices and image memory.

Dynamic microprogramming

The PPP includes seven special purpose modules for two dimensional convolution, point mapping, linear coordinate transformation, logical filtering, histogram generation, region labeling and pixel operations. These functions can be combined for more global and complex functions such as texture analysis, shape identification, region separation and so on. By using dynamically rewritable microprogram architecture, powerful and flexible programs of image processing were designed and demonstrated in the interactive image processing system TOSPICS.[12,13]

IMPLEMENTATION

System configuration

An overall blockdiagram of the PPP is depicted in Figure 2. The host computer in TOSPICS is a minicomputer TOSBAC-40C with 64 K bytes memory. The image memory, the TOSPICS pivot, consists of four layers of 512×512 pixels, 8 bit per pixel, and four planes of 512×512 binary graphic memory. The two dimensional convolution module includes a buffer memory of a shift register and a weighting matrix memory. The shift register storing the image data of n lines enables a parallel accessing of $n\times n$ image data for the local parallel calculations. The 512 byte scratch-pad memory is needed for table look-up operations in logical filtering, point mapping and region labeling. It is also used as 256 registers of 16 bit length in histogram generation.

The image memory is connected to the high speed bus in the PPP, which has a transfer rate of 4 M bytes per second. The interface to the image memory has two address controllers which independently calculate the input and output memory addresses and transfer image data simultaneously to or from any image memory layer or graphic plane. Those address controllers are used for the linear coordinate transformation of image data. They are capable of calculating

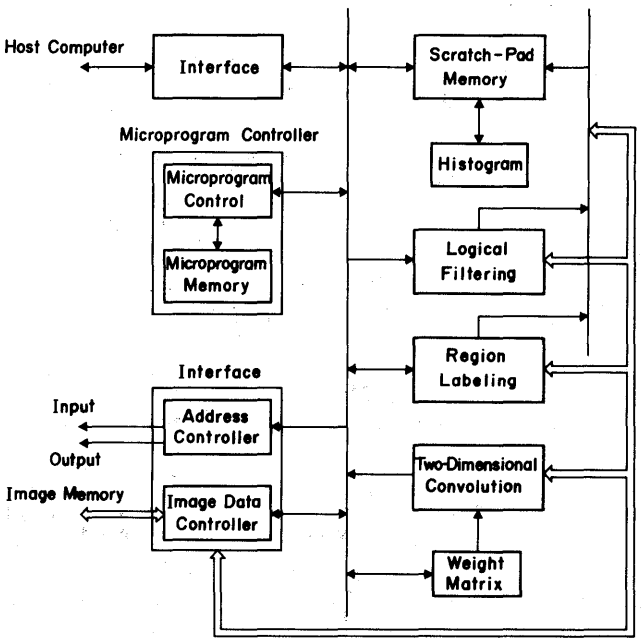


Figure 2—Local parallel pattern processor blockdiagram

next memory addresses automatically, and checking the boundary of image data.

The microprograms and execution commands are transferred through the interface from the host computer with a set of parameters to execute those programs and commands. The processed image data are usually transferred to an image output device, but data can be also stored in the disk memory for the higher level processing, such as image understanding.

Table I shows a repertoire of execution commands. They are grouped in three categories: initialization commands, parameter transfer commands and function execution commands. Initialization commands initiate the internal status of the PPP and transfer any user's microprograms into a random access program memory. Some of the frequently used microprograms, such as enhancement, pixel operations and maximum likelihood decision can be stored in the read-only program memory. Parameter transfer commands permit parameter data communication between the host computer and the PPP. Function execution commands start image processing functions. At the end of operations, an interrupt signal is generated to the host computer.

TABLE I.—Command Repertoire

Mnemonic Code	Function
CLR	PPP Initialization
LMP	Load microprogram in the program memory
LDT	Load data in the scratch-pad memory or in the weight table
SAR	Set parameters in address registers
ASM	Assign memory for processing
TDT	Transfer data from the table memory
FLT	Two-dimensional convolution
AFN	Affine coordinate transformation
HST	Histogram computation
DCV	Data conversion
LFL	Logical filtering
LBL	Region labeling
PIX	Pixel operation

Details of the basic function modulus

Two dimensional convolution module

Figure 3 shows a mechanism for two dimensional convolution function by Eq. (1). It includes m^2 multiplications and (m^2-1) additions for the weighting matrix W_{ij} whose size is $m \times m$. It is obvious that the more parallel arithmetic elements are used, the faster the computation will be performed and the more circuit cost and complexity increase exponentially. To reduce the number of required arithmetic elements, a time-shared pipeline control technique is adopted. The principle of the control is as follows: Eq. (1) is expanded as

$$G(X,Y)=\sum_{j=1}^{m} W_{ij}\,F(X+n-1,Y+n-j) + \cdots + \sum_{j=1}^{m} W_{mj}\,F(X+n-m,Y+n-j) \quad (4)$$

where m is the size of the weight matrix w_{ij}, $F(X,Y)$ is a pixel of input image data at the coordinate values (X,Y), and $n=\left[\frac{m+1}{2}\right]$. By the use of the circuit shown in Figure 4, each term of Eq. (4), a partial sum of products, is calculated by m multipliers and $(m-1)$ adders in parallel, while m terms in Eq. (4) are calculated successively and summed up by the last adder. This configuration of two dimensional convolution module reduces the number of multipliers and adders from m^2 to m, while the total calculation time increases m times, compared with the fully-parallel base configuration.

In practice, the weighting matrix size was determined to be eight as a result of a trade-off between speed and econ-

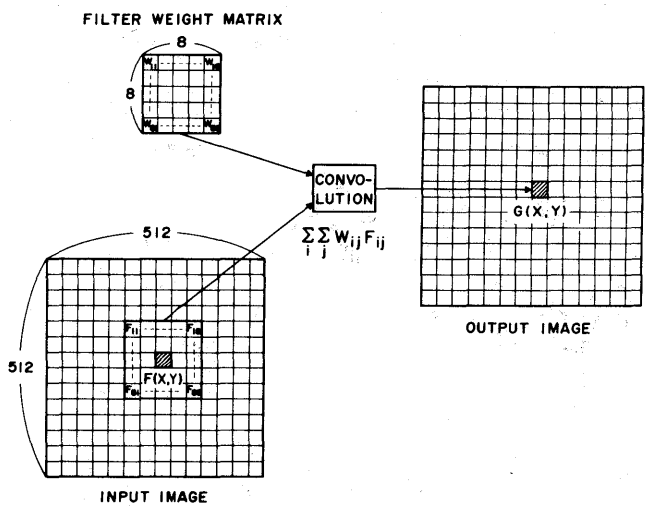


Figure 3—Two-dimensional convolution mechanism

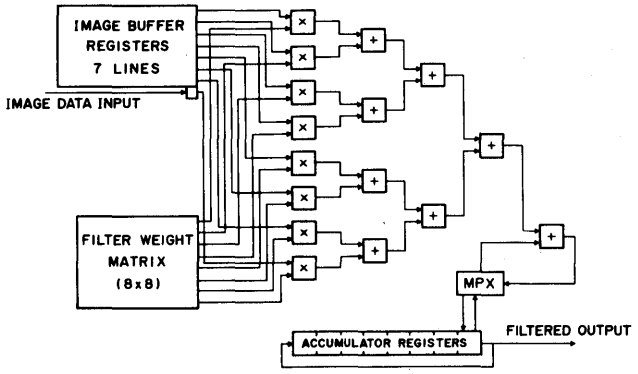


Figure 4—Two-dimensional convolution circuit blockdiagram

omy. Every partial sum of products is calculated at the clock rate of 125 nanoseconds, therefore one micro second is needed to complete the calculation of Eq. (1) for output pixel $G(X,Y)$. This local parallel calculation accomplishes 64 multiplications, 64 additions, and 256 data accesses within 1 micro second. Thus, the input and output data transfer rate between image memory and the PPP is 1 M bytes per second.

In addition, for simplifying data access, a long shift register, which buffers seven lines of image data is used in the two dimensional convolution module, as in Figure 4. A partial column of input image data $F(X+n-1,Y+n-j)$, $j=1,2,\ldots,m$ is used for the calculations of $G(X,Y)$, $G(X+1,Y)$, . . . , $G(X+m,Y)$. If the partial sums of products for those partial column image data in $G(X,Y)$, . . . , $G(X+m,Y)$ are calculated when the column data are shifted at the end of the buffer shift register simultaneously, then the data access time to fetch image data to the arithmetic units can be made nil. Partial sums are sequentially accumulated into the accumulator registers to complete the calculation for output image data.

Logical filtering module

Logical filtering is an operator applied to binary image data for thinning, shrinking, swelling, boundary detection, topological feature extraction operation etc. Figure 5 depicts a result of the logical filtering operation that extracts simultaneously from a thinned binary image, end points (E), branching points (X or Y), line segment directions (1,2,. . . ,8), isolated points (I), and small loop points (0). A logical filtering module takes up every 3×3 submatrix in the image data, and then converts the 9 bit binary data to corresponding output code, consulting the conversion table stored in the scratch-pad memory. The 9-bit binary data are used as a binary address for table look-up of the scratch-pad memory with a 512 byte capacity.

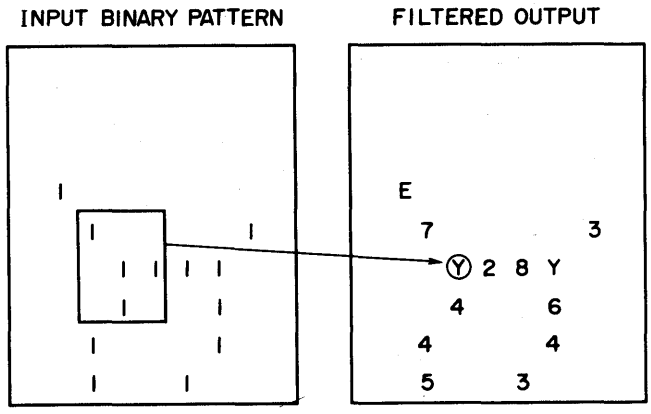


Figure 5—Logical filtering operation result

Linear coordinate transformation module

Two dimensional linear coordinate transformation (affine transformation) is performed by the address control module. Eq. (3) in the earlier section can be rewritten as

$$\begin{cases} X=pX'+qY'+Xo \\ Y=rX'+sY'+Yo \end{cases} \tag{5}$$

where X and Y are the input image coordinates and X' and Y' are the output image coordinates. In digital image processing, image data are expressed in the form of positionally sampled data, so X, Y, X' and Y' are to be integer values. All values in the address control module are expressed in 20 bits with 9-bit fraction. In practice, as the values of X' and Y' change in raster mode, input image coordinates X and Y are calculated by Eq. (5). Since the computed values of X and Y are not always integer, the address control module should interpolate the approximated gray level at coordinates (X,Y) from several surrounding image data, which have integer coordinate values near (X,Y).

Three interpolation algorithms are built in the PPP: nearest neighbor method, bilinear interpolation method and cubic spline function method. The address control module has additional registers to calculate the addresses of neighboring image data points at high speed. Other interpolation algorithms, such as raised cosine method and two dimensional sinc function method, can easily be microprogrammed. Moreover, nonlinear coordinate transformation can be attained by the algorithm using linear approximations for segmented image regions.

The speed of the affine transformation for 512×512 pixels was 0.26 second by the nearest neighbor method, 1.04 second by the bilinear interpolation, and 4.16 second by the cubic spline function method, which uses sixteen neighboring image data points.

Scratch-pad memory and histogram counter

Point mapping of Eq. (2) is done with the scratch-pad memory in Figure 2, where function ϕ is stored in tabular form to be looked up. Because the gray levels of input image data are expressed by 8 bits, the output value corresponding to any input gray level can be stored in the 256 byte capacity scratch-pad memory.

The scratch-pad memory is also used for histogram counting registers for image data. In this case, it sums up registers to count the frequency of points with the same gray level in the input image data specified area. From limitation of the

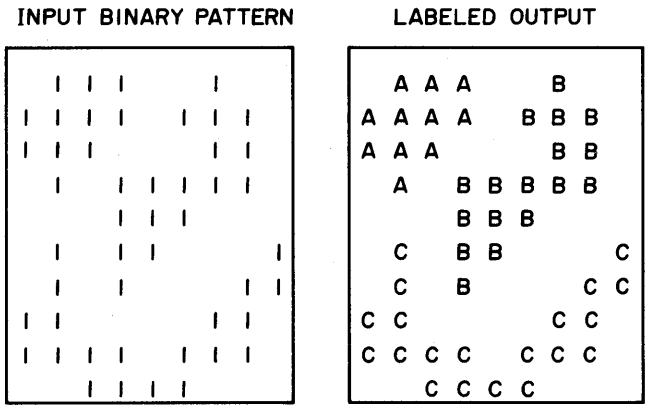


Figure 6—Region labeling operation result

scratch-pad memory capacity (256×2 bytes), HST (Histogram) command is applied to 256×256 image data at a time for a monotonous image. However, it can be applied to 512×512 image data when they are full of variety in gray levels.

Region labeling module

The region labeling module partitions a binary image data into several disjointed blocks by testing the connectivity in a 3×2 submatrix as shown in Figure 6. It is possible to select one of two definitions of connectivity: (a) four direction connectivity, such that if any two "1" points are horizontally or vertically adjacent in image data, after which they are decided to be connected, and (b) eight direction connectivity. In the example of Figure 6, a four direction connectivity mode is applied. If the eight direction connectivity mode is used, regions *A* and *B* would be decided to be connected and assigned by the same label.

Pixel operation module

Beside the most frequently used image processing functions mentioned above, the other problem-oriented operations, such as nonlinear coordinate transformation, maximum likelihood classification and texture analysis are dynamically microprogrammable. Microprograms are transferred from the host computer to the microprogram memory in Figure 2 by LMP command and executed by PIX command. Any arithmetic and logical operations for each pixel are interpreted by the microprogrammable controller (μC). The μC functional roles are as follows:

(1) Pixelwise operation for image data
(2) Communication to the host computer
(3) Interface control to the image memory
(4) Internal data bus control
(5) Control of each functional module.

The word length of a microprogram instruction is 40 bits. The μC cycle time is 250 nanoseconds.

TABLE II.—Applicable image processing repertoire

Basic Function	Applications
Two-Dimensional Convolution	Smoothing, Noise averaging, Enhancement, Differentiation, Restoration, ·····
Affine Coordinate Transform	Enlargement, Shrinking, Rotation, Shift, Geometric Correction, Mosaicking, ·····
Logical Filter	Thinning, Noise Clearing, Gap Filling, Feature Detection, ·····
Region Labeling	Region Separation, Region Coloring, Area Counting, ·····
Data Conversion	γ-correction, log/exp, sin/cos, $x^2/\sqrt{x}$ Thresholding, Histogram Equalization, ·····
Histogram Generation	Gray Level Frequency Counting, Area Counting, ·····
Pixel Operation	Pixewise arithmetic and logical operations, Ratioing, Shading Correction, ·····

APPLICATIONS

The PPP has been proved to be a powerful and economical tool for research and experiments in image processing, especially when it is used in an interactive image processing system. It should be emphasized that the PPP not only has the ability of gray level image processing but also of binary image processing.

A number of subprograms and application programs, using the PPP in TOSPICS, have already been developed for medical image processing, remote sensing and industrial image processing. Table II shows a list of repertoire of applicable image processings by PPP's basic functions. Each basic image processing function can be carried out at the rate of 1 microsecond per pixel. Some examples of execution time required for image data of 512×512 pixels are shown in Table III.

A typical program sequence, using the PPP, would have the following steps in the host computer:

Step 1. Assign image memory for input and output
Step 2. Set parameters in the address controllers

TABLE III.—Some examples of execution time for 512×512 Pixels

Function	Applications	Execution Time
Two-Dimensional Convolution	Laplacian, Smoothing, etc	262mS
Logical Filter	Feature Detection Thinning	262mS 262×W[1] mS
Region Labeling	Region Separation[2] Particle Measurement[3]	524mS 786mS
Data Conversion	γ-Correction Thresholding	262mS "
Histogram Generation	Frequency Counting	262mS
Affine Coordinate Transform	Enlargement with Rotation Axis Skewing	262mS "
Pixel Operation	Logical Operation Arithmetic Operation	524mS "
Gradient Operation	Edge Detection	2.36S

1) W is maximum line width
2) No of separated regions is less than 254
3) Region separation and area counting

Step 3. Specify the basic function or transfer the microprogram string
Step 4. Command execution start
Step 5. Sense the end of execution from the PPP.

The gradient operation to image data, which calculates the magnitude and direction of gradient vectors, is one of the suitable application examples to show an assortment of the basic functions.

If the input image is denoted by $F(X,Y)$, then the magnitude $G(X,Y)$ and direction code $D(X,Y)$ of the gradient vector at point (X,Y) can be obtained as,

$$\begin{cases} G(X,Y)=\sqrt{\left\{\dfrac{\partial F(X,Y)}{\partial X}\right\}^2+\left\{\dfrac{\partial F(X,Y)}{\partial Y}\right\}^2} \\ D(X,Y)=\Theta\left[\tan^{-1}\left\{\dfrac{\partial F(X,Y)}{\partial Y}\Big/\dfrac{\partial F(X,Y)}{\partial X}\right\}\right] \end{cases} \tag{6}$$

where $\partial F(X,Y)/\partial X$ and $\partial F(X,Y)/\partial Y$ are output values of the two dimensional convolution with X-derivative filtering and Y-derivative filtering, respectively. $\Theta(\theta)$ indicates a multilevel thresholding function which encodes an angle θ into one of 16 direction codes.

Let the input image be stored in M1 memory layer, the main stream of subprogram to calculate $G(X,Y)$ and $D(X,Y)$ is as follows:

(a) Execute spatial filtering with X-derivative weighting matrix, and store the results in M2
(b) Execute Y-derivative operation for M1 image data and store the results in M3
(c) Using the point mapping function of x^2, calculate the square of M2 and M3, and add these values to M4
(d) Using pixelwise operation of division, calculate M2/M3 and store the results in M3. Then, execute two point mapping operations, $\tan^{-1}x$ and $\Theta(\theta)$, successively for M3. The direction code $D(X,Y)$ is obtained in M3.
(e) Using point mapping operation of $\sqrt{x}$ for M4, calculate the magnitude of gradient vector $G(X,Y)$ and store results in M2.

The concepts and algorithms of an image processing program for the parallel image processing machine may differ from those of conventional programs implemented in serial digital computers. It is an open problem to develop image processing, such as the relaxation method by Rosenfeld[20] and the primal sketch by Marr.[21]

CONCLUSIONS

A microprogrammable local parallel pattern processor (PPP) has been proposed and realized. The PPP has capabilities of programming local parallel basic operations, coordinate transformations and pixel operations. The processing time is a quarter second for 512×512 pixel image, which is about two orders of magnitude faster than conventional serial computers.

More global image processing functions, such as texture analysis, shape detection, region separation, etc., can be microprogrammed in combination with these basic functions.

Some experimental results have been demonstrated for remotely sensed data and medical image data. These demonstrations indicate that the PPP is sufficiently powerful and economical enough to explore novel applications and quick analysis in the image understanding studies.

A further expansion of the PPP, which is currently being developed, is to realize the fast Fourier transformation and some vector operations in a simple hardware.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Dr. H. Nishino of the Electrotechnical Laboratory and Dr. K. Kakizaki who directed the major phase of this development and encouraged authors, and to Dr. H. Genchi, Dr. S. Watanabe and Mr. T. Kondou who contributed many significant concepts to improving system design. They wish to extend their appreciation to Prof. K. S. Fu of Purdue University for his invitation and encouragement to this session. The research and development are under contract with the Ministry of International Trade and Industry on the Pattern Information Processing System (PIPS) Project.

REFERENCES

1. McCormick, B. H., "The Illinois Pattern Recognition Computer," *IEEE Trans. Electron. Comput.*, Vol. EC-12, No. 12, Dec. 1963, pp. 791-813.
2. Golay, M. J. E., "Hexagonal Parallel Pattern Transformations," *IEEE Trans. Comput.*, Vol. C-18, No. 8, August 1969, pp. 733-740.
3. Gray, S. B., "Local Properties of Binary Images in Two Dimensions," *IEEE Trans. Comput.*, Vol. C-20, No. 5, May 1971, pp. 551-561.
4. Genchi, H., et al., "Recognition of Handwritten Numerical Characters for Automatic Letter Sorting," *Proc. IEEE*, Vol. 56, 1968, pp. 1292-1299.
5. Ingram, M., et al., "Automatic Differentiation of White Blood Cells," Book of Image Processing in Biological Science, ed. by D. Ramsey, Univ. of California Press, 1968, pp. 97-117.
6. Technical Description, "Interactive Multispectral Image Analysis System (IMAGE 100)," GE Space Div. Daytona Beach, Florida, Sept. 1973.
7. Kriegler, F. J., et al., "The MIDAS Processor," *Proc. of 10th Symp. on Remote Sensing of Environ.*, Ann Arbor, Mich., Oct. 1975.
8. Ledley, R. S., et al., "Computerized Transaxial X-Ray Tomography of the Human Body," *Science*, Vol. 186, Oct. 1974, pp. 207-212.
9. Kruse, B., "A Parallel Picture Processing Machine," *IEEE Trans. Comput.*, Vol. C-22, No. 12, Dec. 1973, pp. 1075-1087.
10. Bonknight, W. J., et al., "The Illiac IV System," *Proc. IEEE*, Vol. 60, No. 4, Apr. 1972, pp. 369-388.
11. Rohrbacher, D., et al., "Image Processing with the Staran Parallel Computer," *Computer*, Vol. 10, No. 8, Aug. 1977, pp. 54-59.
12. Mori, K., et al., "Research and Development on Image Processing and Pattern Recognition System," PIPS Project Report, 1977, pp. 69-78 (in Japanese).
13. Asada, H., et al., "An Interactive System for Image Processing—TOSPICS," Paper of Technical Group of Pattern Recognition and Languages of *Institute of Electronics and Communications in Engineering*, Vol. PRL75-51, Oct. 1975, pp. 19-30 (in Japanese).
14. T. Kondou, et al., "Image Processing Software System—TOSPICS EXEC," *Proc. 6th Image Processing Conference*, Nov. 1975, pp. 61-64 (in Japanese).
15. Hunt, B. R., "Computer and Images," SPIE/OSA, *Image Processing*, Vol. 74, 1976, pp. 3-9.

16. Rindfleisch, T. C., et al., "Digital Processing of Mariner 6 and 7 Pictures," *Journal of Geophysical Research,* Vol. 76, No. 2, Jan. 1971, pp. 394-417.
17. Duff, M. J. B., "CLIP4: A Large Scale Integrated Circuit Array Parallel Processor," *Proc. 3rd IJCPR,* 1976, pp. 720-733.
18. Lemkin, P., et al., "A Real Time Picture Processor for Use in Biologic Cell Identification," *The Journal of Histochemistry and Cytochemistry,* Vol. 22, No. 7, 1974, pp. 725-740.
19. Goodenough, D., "A Hybrid System for Image Processing and Pattern Recognition in Remote Sensing," *IEEE Computer Society Conf. on Pattern Recognition and Image Processing,* June 6-8, 1977, Troy, N.Y.
20. Rosenfeld, A., et al., "Scene Labeling by Relaxation Operations," *IEEE Trans. on Syst. Man and Cybern.,* Vol. SMC-6, No. 6, June 1976, pp. 420-433.
21. Marr, D., "Early Processing of Visual Information," AI Memo No. 340, MIT, 1975.

A multi-microprocessor ARES with associative processing capability on semantic data bases

by TADAO ICHIKAWA
Kokusai Denshin Denwa Co., Ltd.
Tokyo, Japan

and

KEN SAKAMURA and HIDEO AISO
Keio University
Yokohama, Japan

INTRODUCTION

Recent progress in architecture technology has had a great impact on the study of pattern recognition problems. The increase in processing power is bringing a new concept of "understanding" into recognition areas. Recognition objectives have been greatly extended to meet real social requirements. As observed in the speech recognition system developed at Carnegie-Mellon University, for example, efforts have been shifted from attaining higher recognition of isolated words to the implementation of a system which provides reasonable understanding of the more natural utterances of connected speech.

This system called HEARSAY[1] provides multi-leveled knowledge sources ranging from parametric representations of the physical features of speech measured every 10ms to the conceptual level of common acceptance. Reliable performances for understanding speech are then attained by the hypothesize-and-test processes at these various levels where these knowledge sources are dynamically related to each other through a general structured global data base.

Concerning images, the University of Massachusetts is developing an image understanding system called VISIONS.[2,3] The system transforms visual sensory numeric data to a symbolic structure representing a model of the image. Conceptual models which are not image-specific are stored as a knowledge source in the semantic data base and serve to specify the observed model by communicating with various levels of features hierarchically extracted through spatial windows of various sizes.

Both of the understanding systems (HEARSAY & VISIONS) are based on the principle of model construction (hypothesizing) and the deductive processing (testing) of conceptual models by the incorporation of a semantic data base (knowledge sources). Thus the associative search of all possible conceptual models on a semantic data base plays an important role in advanced pattern recognition systems, and the efficiency of the search becomes the dominant factor in attaining high performance.

An associative processor ARES proposed by the authors before[4] also provides a powerful tool for associative search on semantic data bases in general because of the ability to associate data items on their deep structures. In the present paper, the association principle of ARES which is based on coding theory is generalized. The association capability is extended by the hierarchical use of a set of different codes. Following the overview of the generalized association mechanism is an architectural description of ARES which is now being developed. By adopting a multi-microprocessor organization, ARES has gained greater adaptability because of the functional flexibility provided by microprogramming.

In conclusion, the multi-microprocessor ARES provides an advanced pattern recognition facility partially bridging the gap between human cognitive behavior and logical processing by computers.

ADVANCED ASSOCIATION FACILITY

A semantic data base is defined generally as a collection of data items which are symbolic representations of speech or image. Data items D's are coded segmentally or categorically as shown in (1) in such a manner that the relatedness of the symbolic partial representations for each segmental or categorical block d_{si} is measured in terms of the distance of the code vectors associated with them.

$$D = (d_{s1}, d_{s2}, \ldots, d_{si}, \ldots, d_{sp}), \quad 1 \leq i \leq p. \quad (1)$$

The length of each segmental or categorical block d_{si} is not necessarily the same depending upon the application.

The association mechanism

Let's divide D into several blocks of equal length n as shown in (2), and apply an error correction scheme of the

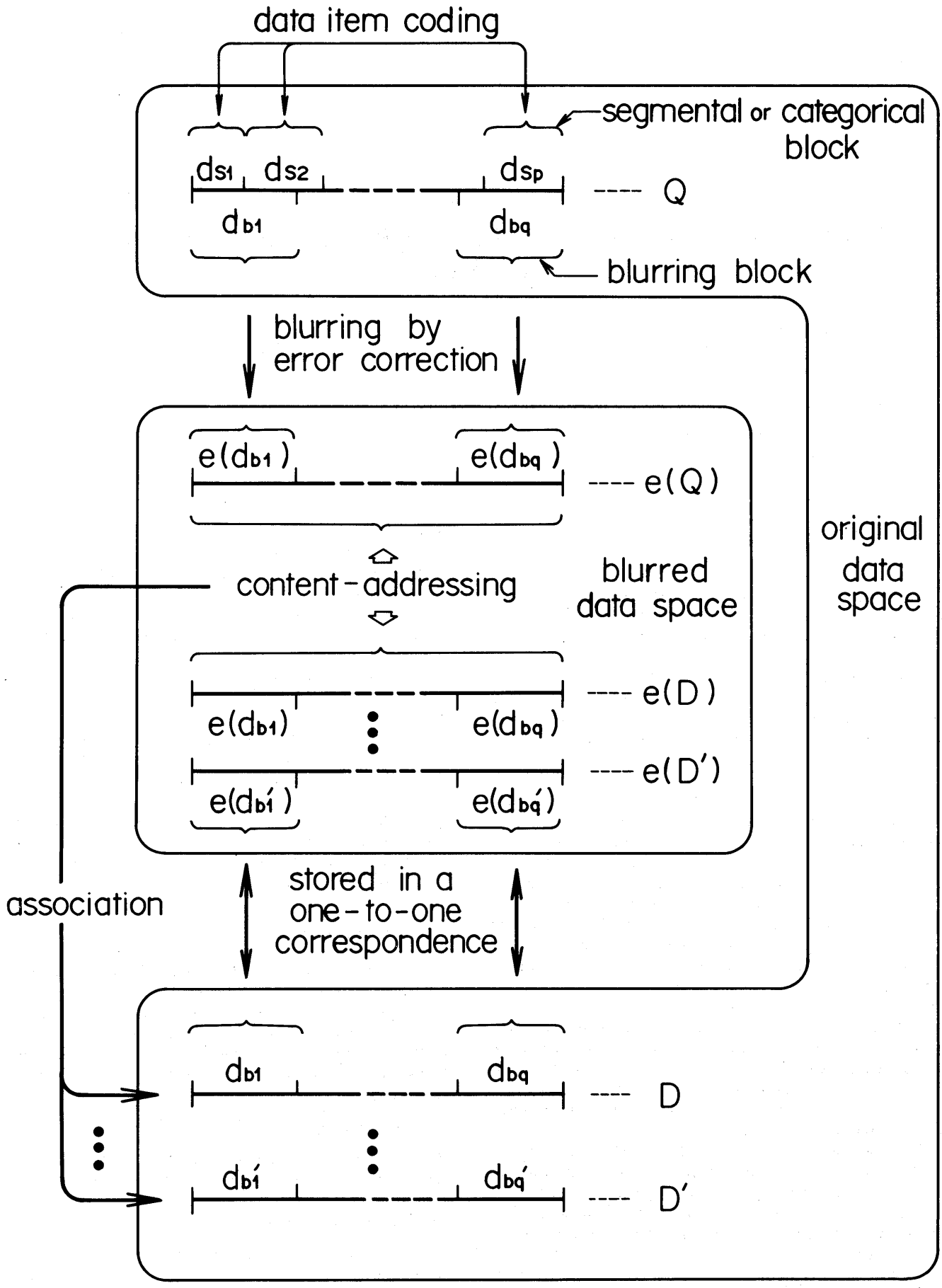


Figure 1—Association process

code of length n with t-error correction capability for each block d_{bj}.

$$D=(d_{b1}, d_{b2}, \ldots, d_{bj}, \ldots, d_{bq}), \quad 1\leq j\leq q. \tag{2}$$

The data representation thus derived is denoted as (3).

$$e(D)=(e(d_{b1}), e(d_{b2}), \ldots, e(d_{bj}), \ldots, e(d_{bq})), \quad 1\leq j\leq q. \tag{3}$$

Each error corrected code vector $e(d_{bj})$ represents a blurred feature of d_{bj}. The d_{bj} is then called a blurring block, and n is normally selected to cover several segmental or categorical blocks d_{si}'s so as to also blur the influence of artificial segmentation or categorization of semantic data items. Thus the $e(D)$ sustains a global structure of data items which are inherently related to each other with equal to or less than t distance.

Now let us apply the content-addressing scheme to $e(D)$'s. When Q is given as a query item, the content-addressing by $e(Q)$ is accomplished by checking the number of blurring blocks which coincide with each other for all $e(D)$'s in parallel, and then $e(D)$'s which coincide with $e(Q)$ in equal to or greater than $\theta(0\leq\theta\leq q)$ number of blurring blocks are selected. Assuming that both D and $e(D)$ are stored as a pair, D's which are strongly related to Q are then specified as associated data items without executing the complex calculations of relatedness.

The space where $e(D)$'s are accommodated is called a blurred data space. Specifying D's through the content-addressing on $e(D)$'s in the blurred data space is referred to as "association". The association process is illustrated in Figure 1.

As explained above, the intuitiveness and ambiguity observed in human cognitive behavior are brought into the association performance of ARES to some extent by connecting the conventional content-addressing and error-correcting technologies, and controlled systematically by the length n and the correctable error distance t of the code applied. Following is a generalization of the blurring by error corrections to reach the more global structure of data items before the association.

Hierarchical process of blurring

Let (n, k, t) denote a code of length n which has k number of information digits with a t-error correction capability, and let the blurred data item derived from D by (n_1, k_1, t_1)-code be represented as $D_1=e_1(D_0)$ where $D_0=D$. Suppose we want to have more general structure of data items, then we apply (n_2, k_2, t_2)-code to D_1 by dividing it into the blurring blocks of length n_2. Thus we can proceed the blurring as indicated in (4) until we reach the blurredness required for the association, where the relation $t_s\geq 2t_{s-1}+1$ should necessarily be satisfied when $n_s\geq n_{s-1}$.

$$D_0\rightarrow D_1=e_1(D_0)\rightarrow D_2=e_2(D_1)\rightarrow\cdots \rightarrow D_s=e_s(D_{s-1})\rightarrow\cdots. \tag{4}$$

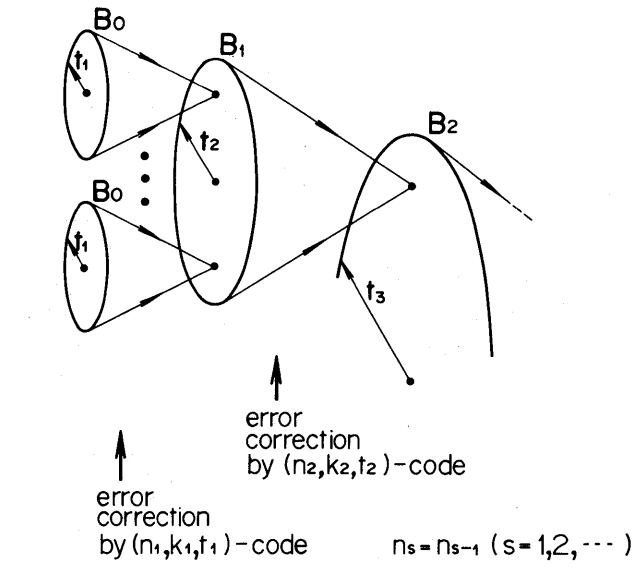


Figure 2—Blurring process

Let the original data space and blurred data space of level s be denoted as B_0 and B_s, respectively. Data items in B_0 are strongly dependent on physical features directly measurable at B_0. Data items in B_s, however, represent more general features commonly possessed by a larger class of data items, and the features go up to the conceptual or semantic level as the blurring proceeds under the hierarchical application of error corrections.

Perfect codes are desirably selected for the blurring. Figure 2 illustrates schematically the hierarchical blurring of a single block of coded data items for the case of $n_s=n_{s-1}$ $(s=1, 2, \ldots)$. Blocks of length n_s located in a circle of radius t_s in B_{s-1} are merged into one in B_s since they must have been strongly related to each other.

Association based on a deep structured relatedness

Content-addressing for the association can be applied at any level of blurring when the feature provided is satisfactorily general to meet the requirements. Let s^* denote the blurring level where content-addressing is applied. Then D_0's in B_0 to be associated are directly specified together with the related D_s's $(0<s<s^*)$ when necessary, without tracing back the blurring processes to B_0 since D_s $(0\leq s<s^*)$ and D_{s^*} are stored in a one-to-one correspondence.

Figure 3 is a schematic diagram explaining how the content-addressing and hierarchical blurring are connected for performing association. The query item Q_0 (ϵB_0) might in some cases be applied externally. The hit marks indicate the D_{s^*}'s which are content-addressed in B_{s^*}. When we have a restriction on the number of associated data items, the

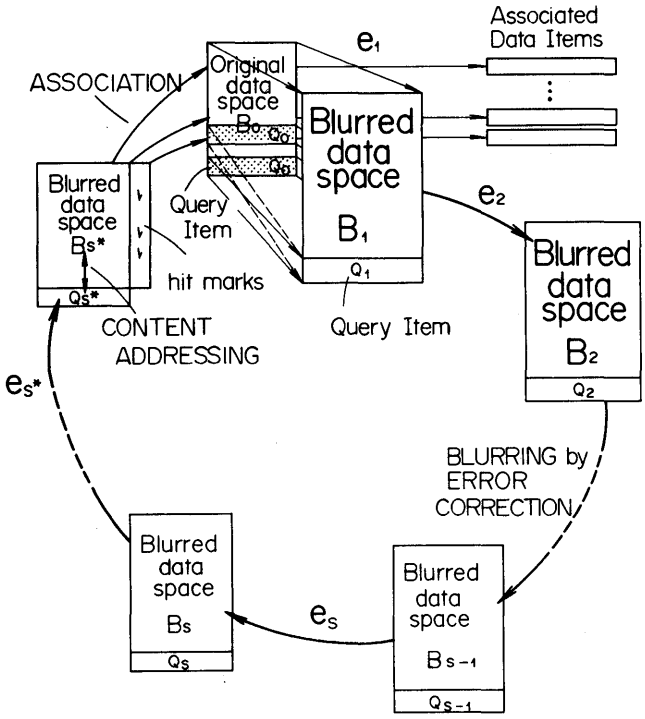


Figure 3—Association through blurring and content-addressing

threshold θ for content-addressing is heuristically selected so that hit marks do not exceed a certain limit δ.

ARES has data modification facility. Suppose Q_0 (ϵB_0) is given as a query item. If some D_0's in B_0 are specified by Q_0 through the content-addressing at the blurring level s^*, these D_0's and related D_s's ($0<s\leq s^*$) are replaced by Q_0 and its blurred expressions, respectively. Otherwise, Q_s ($0\leq s\leq s^*$) is registered, unmodified, to B_s ($0\leq s\leq s^*$). Q_0 might be given as a weighted average of two data items previously stored in B_0.

A MULTI-MICROPROCESSOR IMPLEMENTATION OF ARES

In order to effectively carry out the associative search on a semantic data base described in the previous sections, the following functions should be provided at the computer architecture level:

(1) A hierarchical blurring capability.
(2) The fundamental content-addressing capability.
(3) A heuristic control mechanism to obtain a reasonable number of associated data items.

For the hierarchical blurring capability, a control logic is

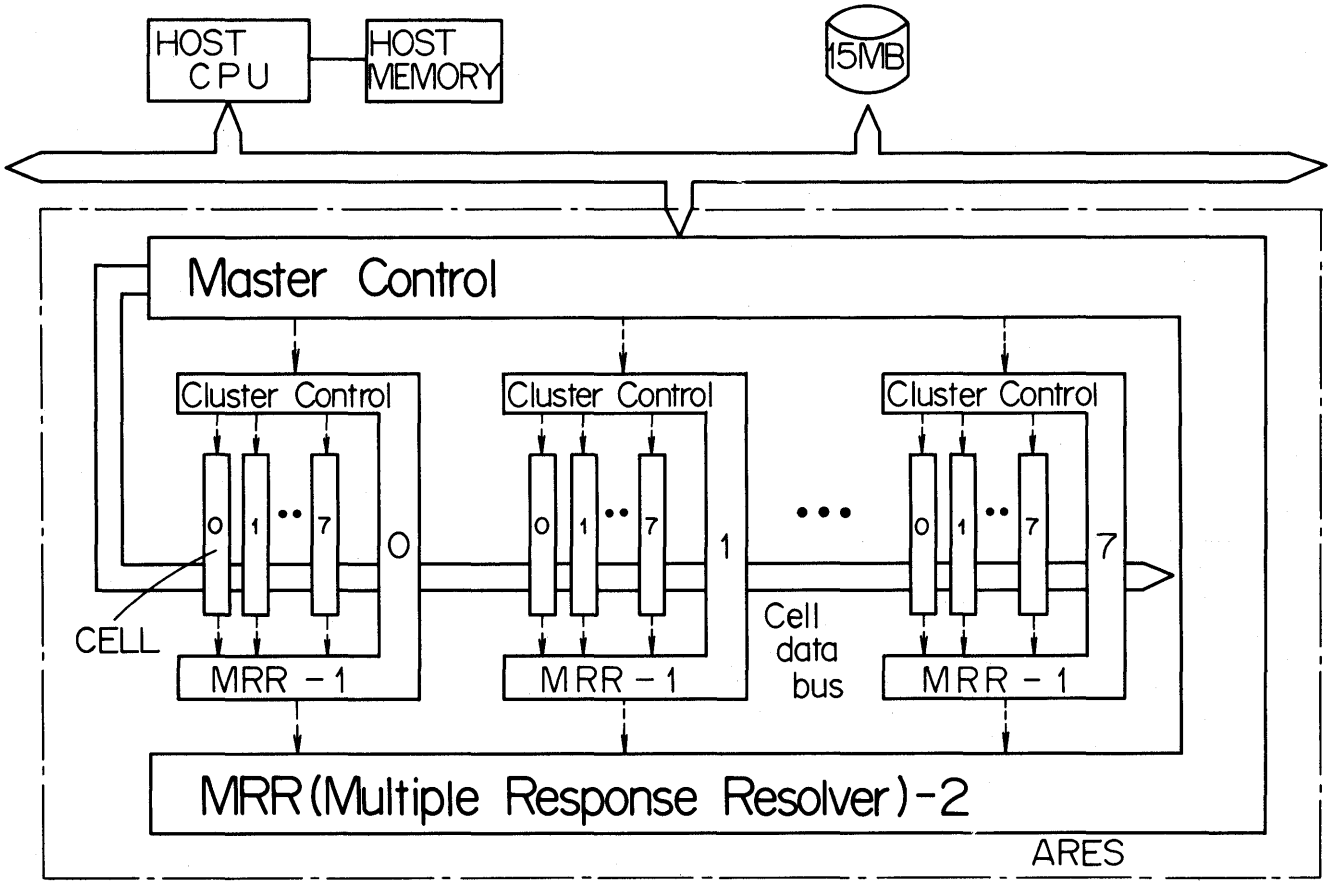


Figure 4—ARES main block diagram

necessary which offers an efficient access to the multiple blurred data spaces and executes the error correction schemes applied for the blurring. Binary cyclic codes are useful for a variety of applications because of the flexibility of selecting proper length and the easiness of implementing the error correction procedure.

As shown before in Figure 3, the content-addressing is accomplished in the blurred data space B_{s*} taking each blurring block of the final stage of blurring as a unit, and for all units in parallel. The number of blocks which coincide with each other is counted over all the blocks unmasked, and compared with a certain predefined threshold value θ. When it is equal to or greater than θ, the corresponding hit mark is set to 1, and the value of θ is gradually adjusted by the heuristic control mechanism embodied in ARES so that a greater number of associated data items as possible are obtained within the limit δ.

In principle, these functions associated with the content-addressing can be accomplished by the mechanism which is incorporated in the uniprocessor ARES originally designed for the recognition of handwritten characters.[4] However, in order to facilitate the blurring time after time and to effectively create the blurred data spaces, it is extremely desirable to provide a logic which works over many sets of blocks in data words simultaneously. Furthermore, a blurred data space should be spatially distributed again in order to execute the blurring and content-addressing for all data words in parallel. These concepts lead to a conclusion that a multiprocessor organization is the best of possible approaches to attain the advanced association facility described above.

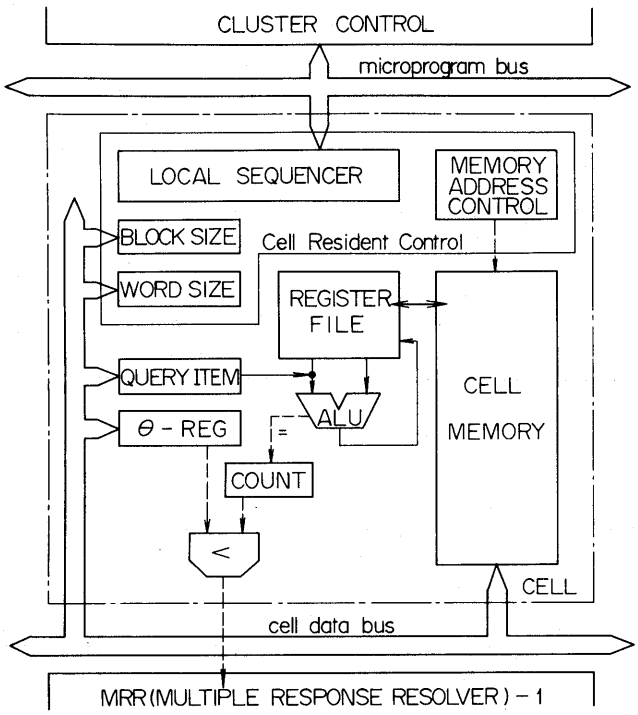


Figure 5—Cell diagram

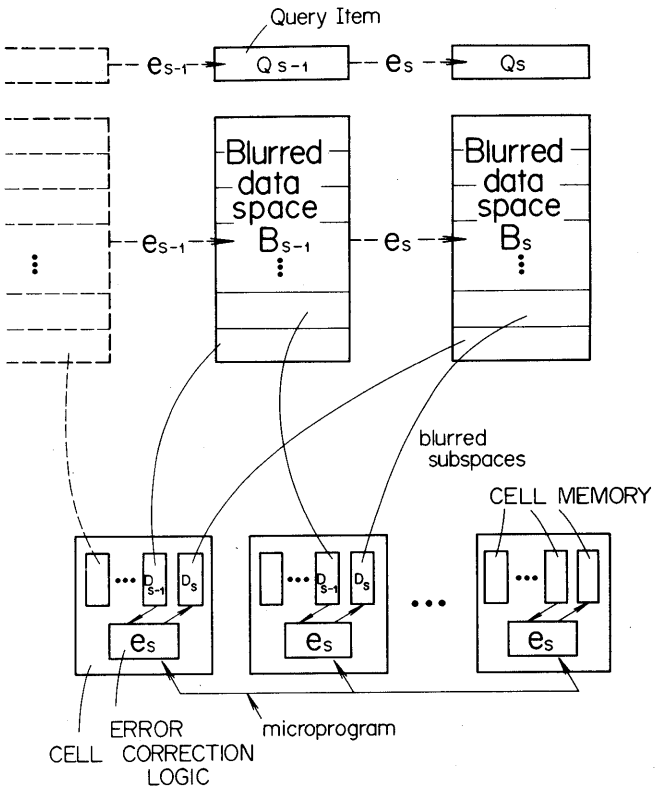


Figure 6—ARES in blurring mode

Architecture of ARES

Figure 4 shows the block diagram of multiprocessor organization of ARES. The distributed processing requests the Master Control to synchronize the whole logic units. As the logic unit, a functionally flexible logic, for instance, a microprocessor (TI-SN74S481; 16Kbytes memory) is used since the data structure of semantic data bases might be considerably complicated depending upon the problems to be solved. Hereafter, the logic unit is referred to as a cell.

The Cluster Control controls eight cells all together, and the master control governs eight cluster controls. Therefore, up to eight different programs may be processed in eight different clusters of cells in parallel. On the other hand, in the case that all the cluster controls are assigned to process the same program, sixty-four parallel operations can be performed in the whole cells.

The Multiple Response Resolver (MRR) which is a unique facility in ARES is made up of a parallel counter with multiple inputs. Each MRR-1 receives the hit marks from the cells attached to it, and the contents of MRR-1's are collected by MRR-2. The master control compares the content of MRR-2 with the predefined value of δ and adjusts the value of θ when necessary for further content-addressing. The design of MRR is basically made clear in Reference 4.

Figure 5 shows the internal structure of a cell which is the nucleus of the association logic. A data item and its blurred

representations are accommodated in a cell together with the logic encountered in a conventional associative memory. The block size and the word size indicate the length of a blurring block at each stage of blurring and the length of a data item, respectively.

We may conclude that the clustering of cells supplies high reliability and functional flexibility; the loss of a single cell function or MRR-1 results in a slight decrease of processing power but no complete breakdown of the system, and each cluster of cells can perform its own function individually.

Functional behavior of ARES

The functional behavior of ARES for manipulating a semantic data base is explained as follows: The mini-computer HP-21MX is used as a Host Computer, and recognizes ARES as an associative memory with extremely high intelligence. The operation is initiated when the host computer issues a global command to the master control. The master control analyzes the command and gives the related subcommands to the cluster controls. This implies the function of pointing out the start addresses of the microprograms to be performed, which are previously stored in the cluster controls. Then each cluster control carries out the specified operation individually.

Figure 6 shows the relationship between the blurred data spaces and the corresponding cells in the blurring mode.

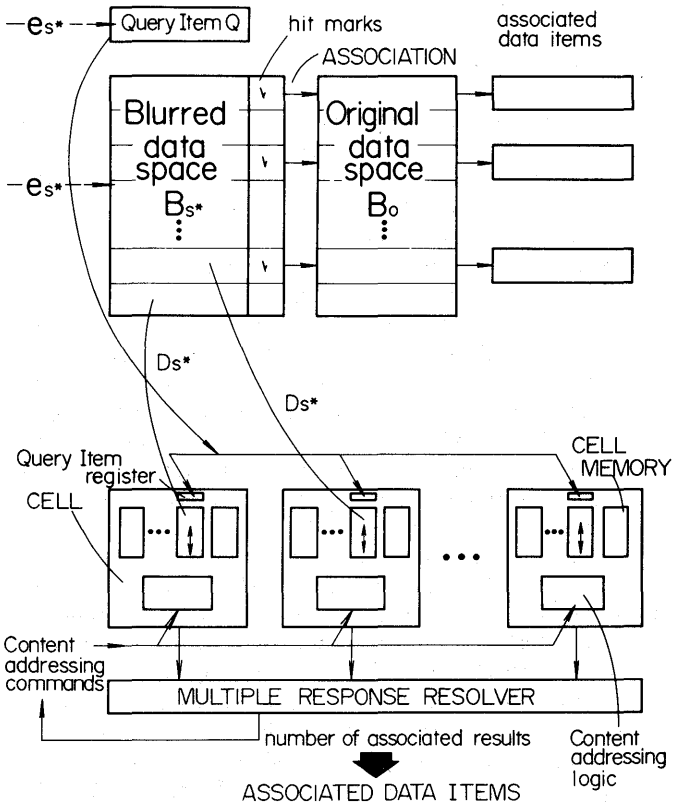


Figure 7—ARES in content-addressing mode

Each blurred data space is divided into subspaces and data items contained in a space are distributed to the cell memories so that they can be processed in parallel. The error correction for the blurring is accomplished through ALU shown in Figure 5, and controlled by the microprograms in the cluster control.

Figure 7 indicates the same relationship in the content-addressing mode. A query item and the value of δ are sent to each cell in parallel by the master control. The content-addressing is then performed in the blurred data space B_{s^*} which are distributed in cell memories, with θ heuristically controlled by MRR under the parameter δ. The control of θ is implemented in hardware because the very fast multiple associations are required repeatedly. The hardware is designed in consideration of the trade-off between the functional requirements imposed and the cost of implementation.

CONCLUSION

In this paper, the association principle presented before[4] is extensively developed in order to be applied to the associative search on semantic data bases for speech or image understanding. Data items are coded in such a manner that the relatedness of the symbolic partial representations of segmental or categorical blocks of data items is measured in terms of the distance of the code vectors associated with them, and they are blurred by the hierarchical application of error correction schemes. Thus, by simply applying conventional content-addressing schemes to the blurred data items, the associative search of original data items is accomplished on their deep structured relatedness without executing the complex calculations of relatedness on original data representation.

Following the overview of the generalized association mechanism, we presented an architectural description of ARES which is now being developed. From the viewpoint of computer architecture, ARES has the following characteristic features: A multi-microprocessor organization is adopted using HP-21MX mini-computer and TI-SN74S481 Schottky I^2L bipolar microprocessors as a host computer and cell components, respectively. This provides both the structural extendibility and functional flexibility supported by microprogramming to adapt to a variety of applications in the field of artificial intelligence. Furthermore, ARES facilitates heuristic control of the number of associated data items, which is useful to solve pattern recognition problems.

So far we have discussed the associative search on semantic data bases for speech or image understanding. However, ARES can be well applied to natural language processing. For example, the associative-categorical model proposed by Haralick and Ripken[5] for semantic word retrieving is dealt with by means of representing categorical relatedness of data items by code distances.[6]

In conclusion, a multi-microprocessor ARES provides an advanced pattern recognition facility bridging the gap between human cognitive behavior and the logical processing by computers.

ACKNOWLEDGMENT

The authors wish to thank Professor C. V. Ramamoorthy of UC, Berkeley for his encouragement.

REFERENCES

1. Lesser, V. R., R. D. Fennel, L. D. Erman, and D. Raj Reddy, "Organization of the Hearsay II Speech Understanding System," *IEEE Transaction on Acoustics, Speech, and Signal Processing,* Vol. ASSP-23, No. 1, February 1975, pp. 11-24.
2. Hanson, A. R. and E. M. Riseman, "The Design of a Semantically Directed Vision Processor (Revised and Updated)," *COINS Technical Report 75C-1,* University of Massachusetts, Amherst, February 1975.
3. Hanson, A. R. and E. M. Riseman, "A Progress Report on VISIONS: Representation and Control in the Construction of Visual Models," *COINS Technical Report 76-9,* University of Massachusetts, Amherst, July 1976.
4. Ichikawa, T., K. Sakamura, and H. Aiso, "ARES—A Memory, Capable of Associating Stored Information through Relevancy Estimation," *AFIPS Conference Proceedings,* Vol. 46, June 1977, pp. 947-954.
5. Haralick, R. M. and K. Ripken, "An Associative-Categorical Model of Word Meaning," *Artificial Intelligence,* 6, 1975, pp. 75-99.
6. Ichikawa, T. and M. Christiansen, "On a Semantic Word Retrieving," *Proceedings of the National Convention of IECE-Japan,* March 1978, p. 1167 (in Japanese).

The STARAN architecture and its application to image processing and pattern recognition algorithms

by J. L. POTTER
Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

The STARAN E is a general purpose parallel computer. However, certain aspects of the STARAN E's architecture, specifically, a single instruction stream-multiple data stream organization, high speed I/O, a flip (permutation) network and a conditional operation capability on each parallel processing element, are particularly pertinent to the areas of image processing and pattern recognition.

Some of the characteristics of image processing and pattern recognition which provide a good fit to the STARAN's architecture are repetitiveness, spatial dependencies, complex parallel decision making and high I/O to computation ratios. These characteristics and their corresponding STARAN E architectural accommodations are described in detail. The other aspects of STARAN's architecture are described in only enough detail to provide a basis for discussion.*

BACKGROUND

The STARAN E consists of an associative processor (AP) control module and a number (1 to 32) of associative arrays. The AP module consists of the AP control circuitry itself and bulk core. Figure 1 shows the basic STARAN architecture.

The arrays can be thought of as consisting of high speed memory, low speed memory and a bank of processing elements. Each array is organized into 256 "words." Associated with each "word" is a processing element (PE). Each word is 9K bits long with 1024 bits of high speed memory and 8192 bits of lower speed memory. Figure 2 illustrates the conceptual array organization and a general purpose layout for a 512×512 8 bit/pixel image.

Arithmetic operations are performed in parallel on every (enabled) word of memory, one bit at a time. That is, the least significant bit of every word in a field (a bit column vector) is added (multiplied, etc.) to the corresponding least significant bit of every word in a second field and stored in the least significant bits of the sum (product, etc.) field. The carry bits are saved and added into the second bit slices of the arguments. This process is repeated until the entire fields have been processed.

ARCHITECTURE—ALGORITHM PARINGS

Certain characteristics of image processing and pattern recognition mesh perfectly with some of the architectural features of the STARAN E. In the following paragraphs,

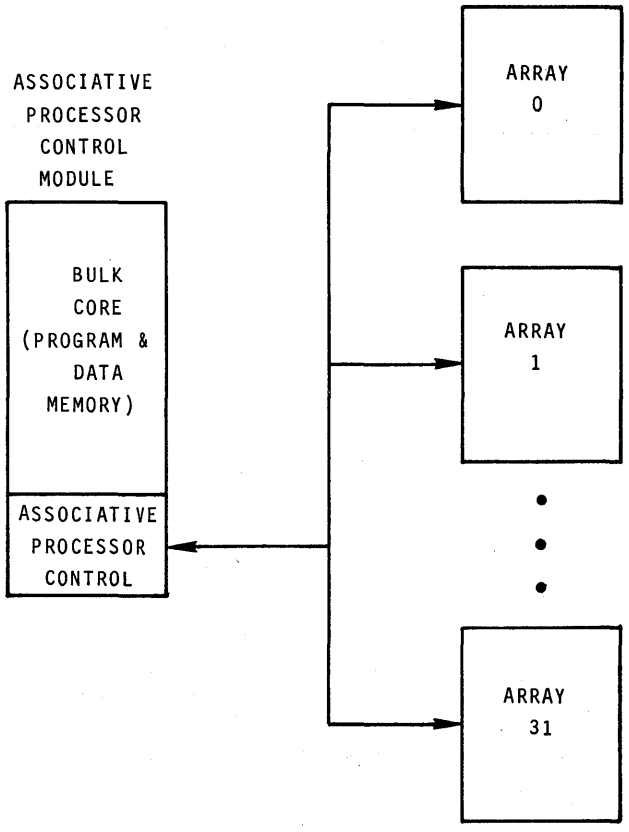


Figure 1—Basic STARAN architecture

* For a detailed description of the STARAN line of computers see References 1, 2 and 3.

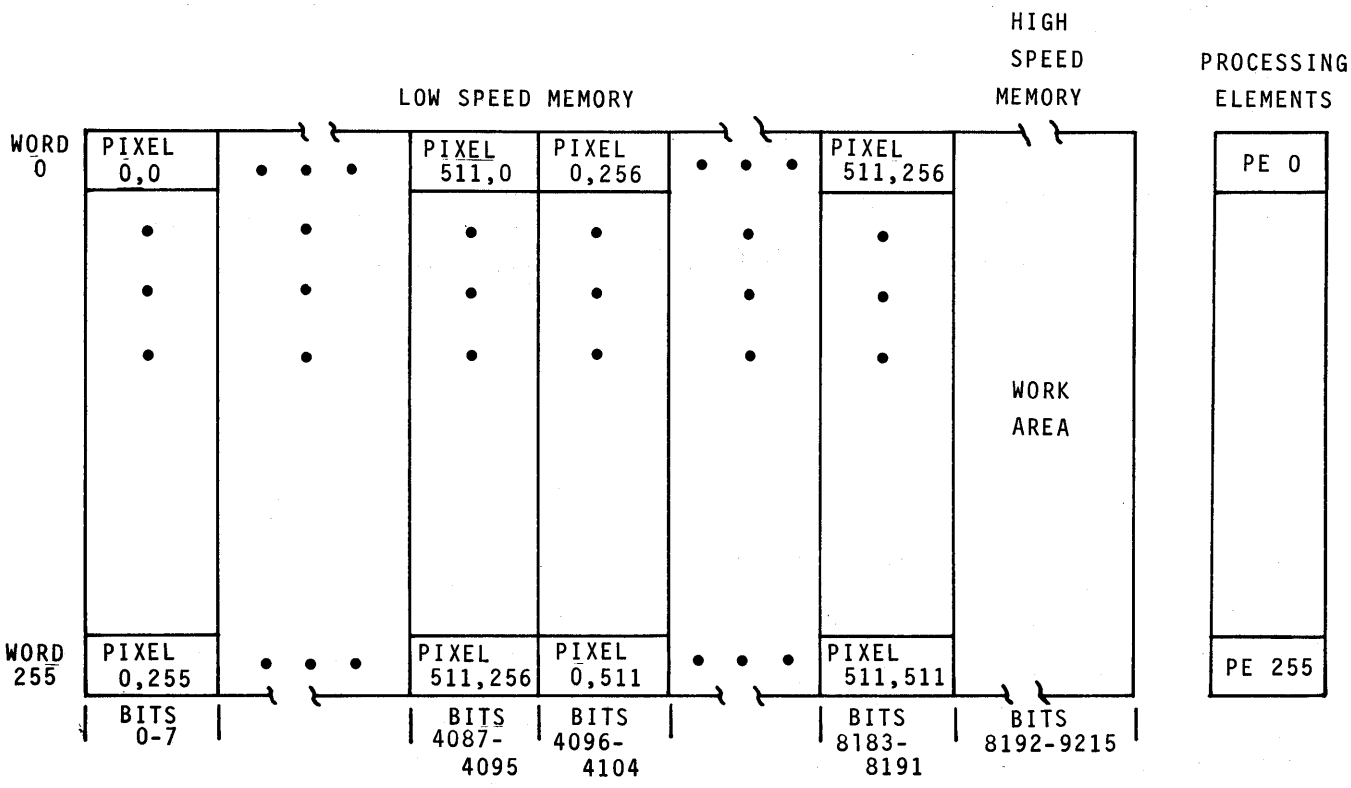


Figure 2—General purpose array organization

some characteristics of image processing and pattern recognition algorithms will be identified and described. Then the corresponding architectural feature of the STARAN E will be discussed and shown how it facilitates implementation of the algorithms.

REPETITIVENESS

Images consist of large volumes of data. A standard TV monitor size image contains over a quarter of a million pixels (picture elements). The images obtained from satellites may contain many millions of pixels. Almost all image processing and pattern recognition algorithms consist of performing the same sequence of operations for every pixel in an image. This aspect of image processing fits perfectly with the single instruction stream-multiple data stream organization of the STARAN.

The associative processor (AP) control portion of the STARAN computer provides a single sequential instruction stream to the associative arrays. Each associative array contains 256 processing elements (PE's) and a fully equipped STARAN E may have up to 32 arrays. Thus the single instruction stream can control from 256 to 8192 PE's resulting in a processing capability of from 11 to 356 million 32-bit adds per second (MIPS). Figure 1 illustrates the single instruction stream-multiple data stream organization of STARAN.

"Inherently serial" algorithms such as classical Maximum Likelihood classification are easily implemented in parallel with the above organization. This is because the algorithm is still executed in serial for every pixel, but from 256 to 8192 pixels can be handled with one pass of the algorithm. With the large number of pixels that need to be processed in a typical image, parallel application of the algorithm is the only practical answer.

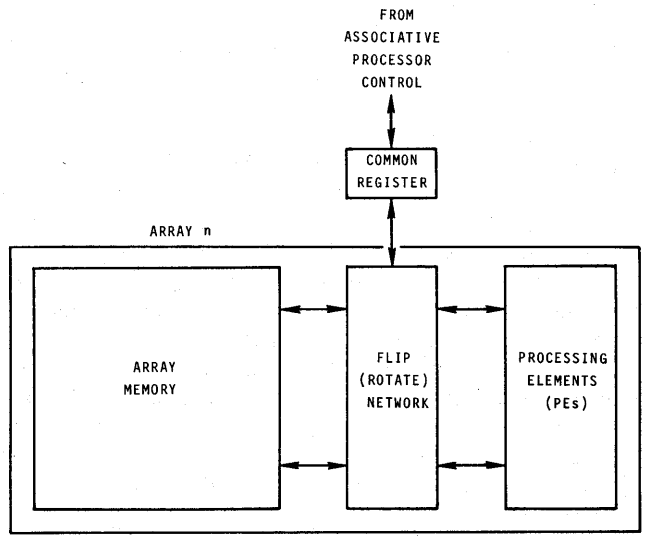


Figure 3—Flip network

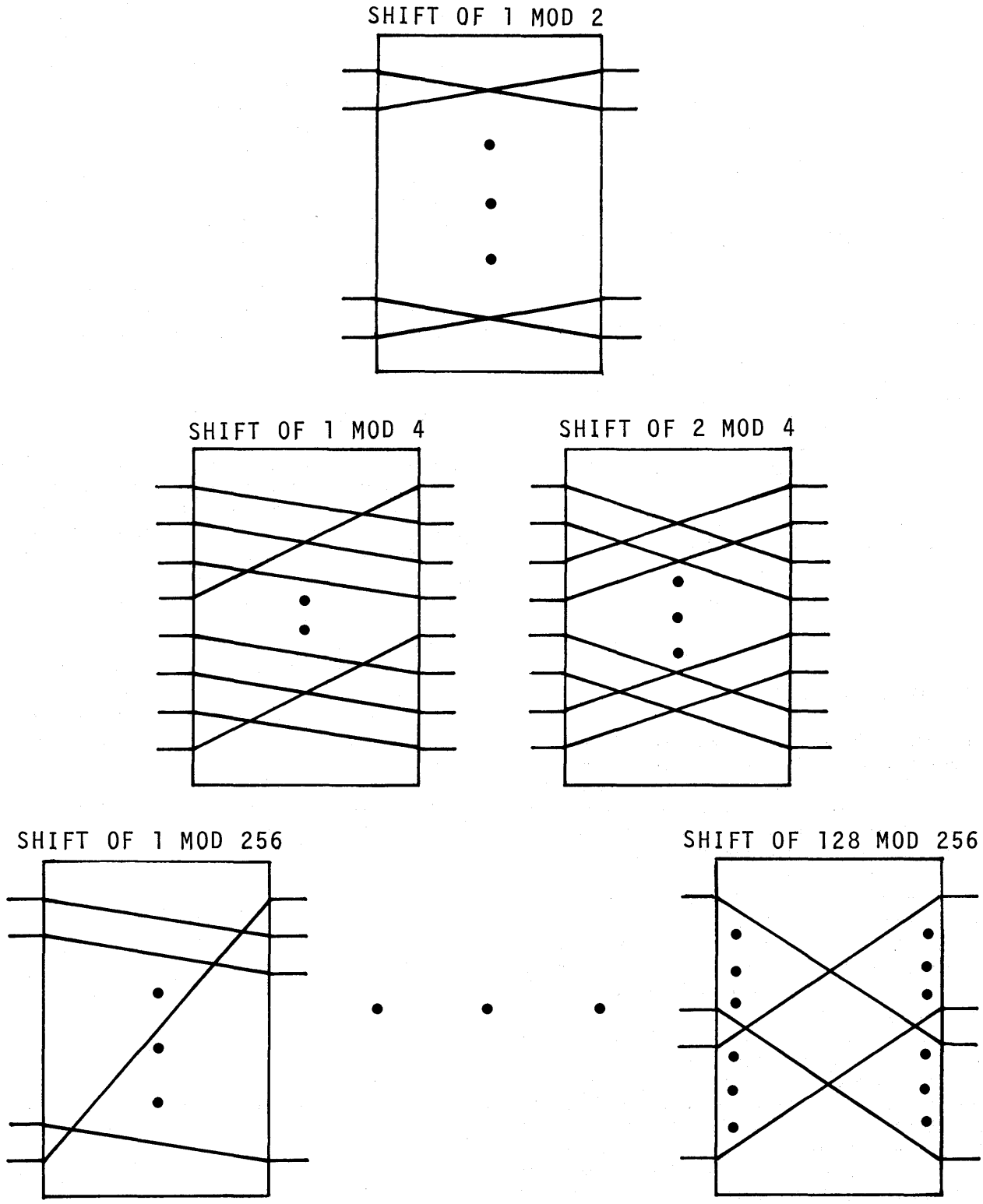


Figure 4—Flip network rotations

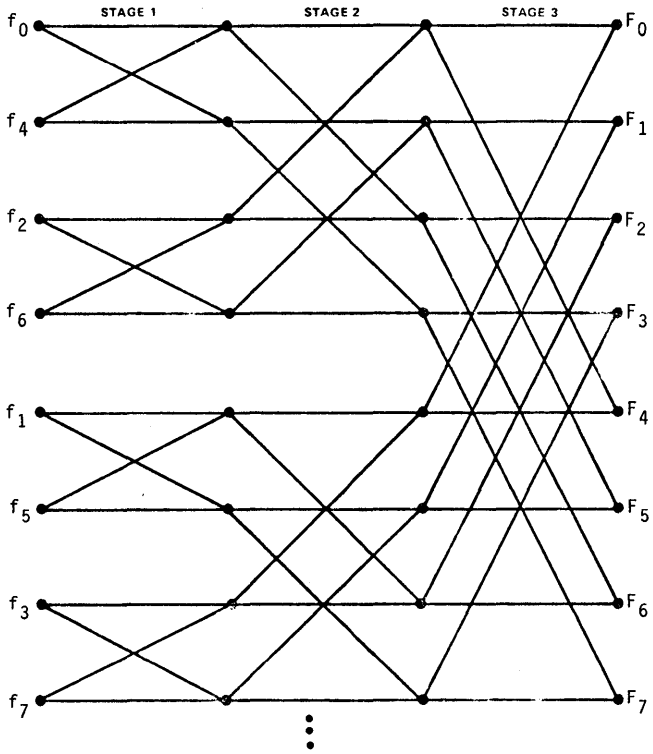


Figure 5—Butterfly diagram for FFT

SPATIAL DEPENDENCIES

Image data is inherently two dimensional and processing algorithms often must deal with this "built in" organization. In general, there are two different types of relationships. The neighborhood relationship requires that pixel neighbors in the two dimensional plane be readily accessible. The second type of relationship is normally spatially more extensive and is typified by the Fourier Transform where the pixel and its neighbors at power of two intervals for an entire row or column are related.

The STARAN associative arrays are well suited for accommodating both types of dependencies. Each array, as shown in Figure 3, has a flip network located between the memory and PE portions. This network performs as a specialized shift register and provides great flexibility in accessing pixels which are spatially related.

In particular, the flip network can accomplish power of two rotates within power of two sized fields with no time penalty. That is, a 1 bit rotate in 128 2 bit fields; 1 and 2 bit rotates in 64 4 bit fields; 1, 2 and 4 bit rotates in 32 8 bit fields; on up to 1, 2, 4, 8, 16, 32, 64 and 128 bit rotates in one 256 bit field. (See Figure 4) This means that inter-word operations at these intervals can be performed in parallel at the same rate as intra-word operations. Moreover, all inter-word operations can be performed with only a slight penalty.

The flip network then provides a very efficient method of implementing algorithms such as the Fast Fourier transform which utilize the spatial power of two interrelationships between pixels. This power of two spatial relationships is frequently expressed in the butterfly diagram shown in Figure 5. A detailed discussion of how the FFT can be implemented in the STARAN in log N steps of 1 add, 1 subtract, 2 real multiples and 2 exchanges has been published elsewhere.[4]

Template matching and spatial convolution are two algorithms which require the neighborhood type of pixel access. These processes can be implemented with little or no time penalty for the required spatial relationships. In particular, 2×2, 3×3, 5×5, 9×9 and 17×17 displacements require no time penalty. Other displacement amounts up to 16×16 require at most one extra shift except for 12×12 and 14×14 which require two shifts. In general, the required shifting for processing *any* window or template size within the 256 word array (i.e. 255×255 or less) is insignificant in overall algorithm time.

PARALLEL DECISION MAKING

An important aspect of image processing and pattern recognition is decision making. Many algorithms perform different operations as a function of the data. The complexity of the process is typified best perhaps by scene analysis techniques. In these situations it is essential to be able to record the exact state of each individual datum in the image.

Each STARAN array contains a special register which can be set as the result of searches (LE, GT, etc.) and arithmetic operations. This mask (or M) register can be used to select a subset of the words in an array to participate in subsequent operations. The results of tests, conditions and states can be stored, retrieved and operated on to achieve any desired logical combination of tests. Figure 6 indicates the physical location of this register.

Two examples of how the M register can be used are template matching and hierarchical structuring. To start a 3×3 template match, the M registers are set to all ones so

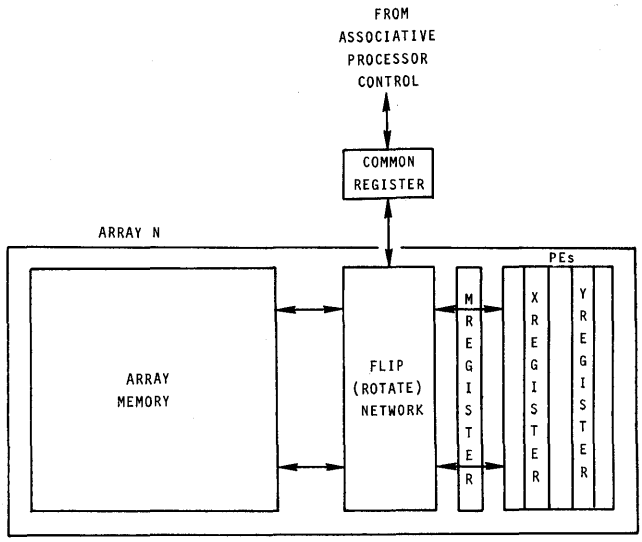


Figure 6—M register configuration

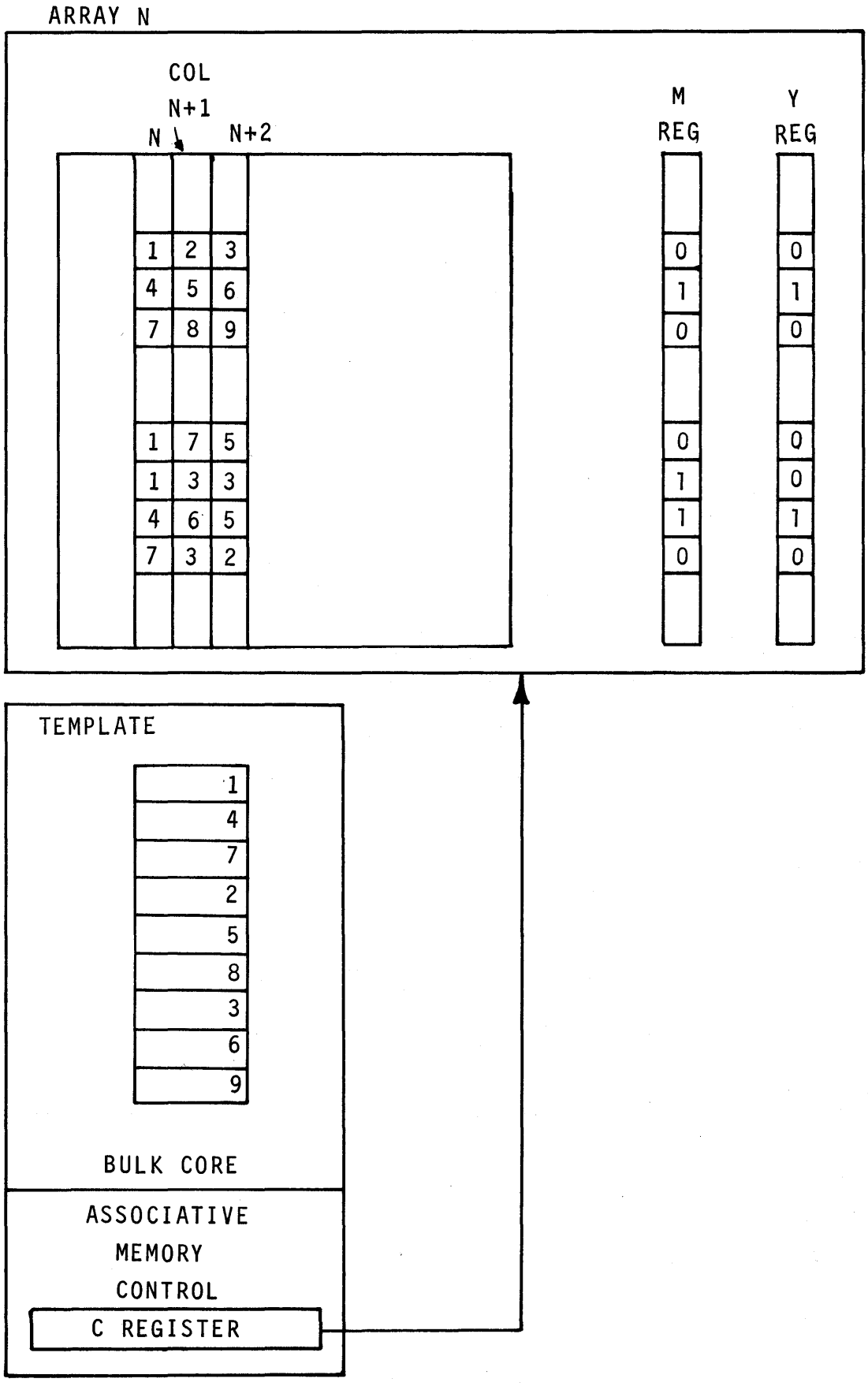


Figure 7—Template matching

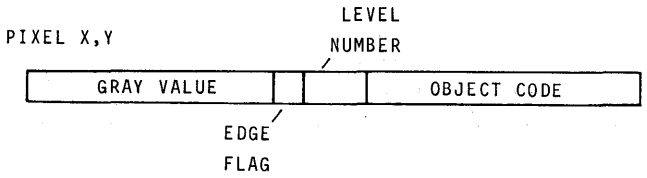


Figure 8—"Typical" scene analysis configuration

that every word participates in the search. The first value of the template is loaded into the common register and the first columns of all the arrays are matched against it. Every successful match is recorded in the Y register. The Y register is shifted down one and moved to the M register, the second template value is moved to the C register and the column is searched for a match of the second template value but on only those words with a corresponding one in the M register (i.e., only those which successfully matched the first value). Consequently at the end of the second search the Y register contains a one for every word and only those words which successfully matched both the first and second template values (the state shown in the M and Y registers in Figure 7). The procedure is repeated for each of the template values with an upward shift and column increment after searching for the third and sixth values. At the end of this process, the M register of all enabled arrays will contain a one (shifted down by two) corresponding to every successful template match covering columns 1, 2 and 3. The process is repeated for every set of columns to be searched.

Note that the columns need not be contiguous to be searched. Thus two situations can be easily handled. First, the columns can be organized in an order which facilitates another algorithm and/or input/output situations. Second, the template need not be contiguous but may cover essentially any size area in any desired manner. Both of these capabilities emphasize the flexibility of data processing in the STARAN arrays.

In a scene analysis application, each pixel might have a set of flags and auxiliary fields associated with it as shown in Figure 8. Then searches of the type "obtain the edges of object 5 on level 2" would be easily implemented in parallel by: first, searching all pixels in a column for object 5, then searching the matched words for level number 2 and then ANDing the edge flags with the Y register. The result is the answer to the search for the given pixel column. The process would then be repeated for each column under consideration. This example illustrates how easy it is to save the result of previous algorithms as flag vectors and codes in parallel and that this information is readily available for subsequent processing and analysis.

INPUT/OUTPUT VERSUS COMPUTATION

It has been established that in general, image processing involves large volumes of data. The algorithms vary quite markedly however in the degree of computation involved. Simple grayscale remapping is such a useful function that special hardware circuitry is often contained within the display devices. In order to make such changes permanent, however, a computer must be used. Operations such as changing (remapping) the grayscale values of images require only one operation per pixel but must be performed on every pixel. These algorithms represent the high I/O-low computation end of the spectrum. At the other end are such algorithms as the domain transforms. Frequently these procedures require numerous arithmetic operations on a per pixel basis. For example, a two dimensional Fast Fourier Transform for a 512×512 pixel image requires 54 multiples and 90 adds per pixel (for a serial computer).

The STARAN has three I/O paths. The common register path (shown in Figure 1 and at the top in Figure 9) operates at between 12-15 million bits per second (MBPS). It is most useful for "broadcasting" data such as constants and parameters to all arrays in parallel. It is a 32 bit wide path.

The most useful path for data I/O is the 32 bit wide multiplexed I/O bus into and out of each array. This bus is capable of operating at between 80 and 640 MBPS. Thus an entire 512×512 8 bit per pixel image can be input or output in from 26.2 to 3.3 milliseconds. This data path is connected to a crossbar switch so that it can be used to transmit data between arrays as well as to peripheral storage or image display devices.

The fastest bus is the 256 bit parallel I/O bus which can operate from 512 to 2560 MBPS. The data transfer rate on this bus is such that special peripheral configurations are

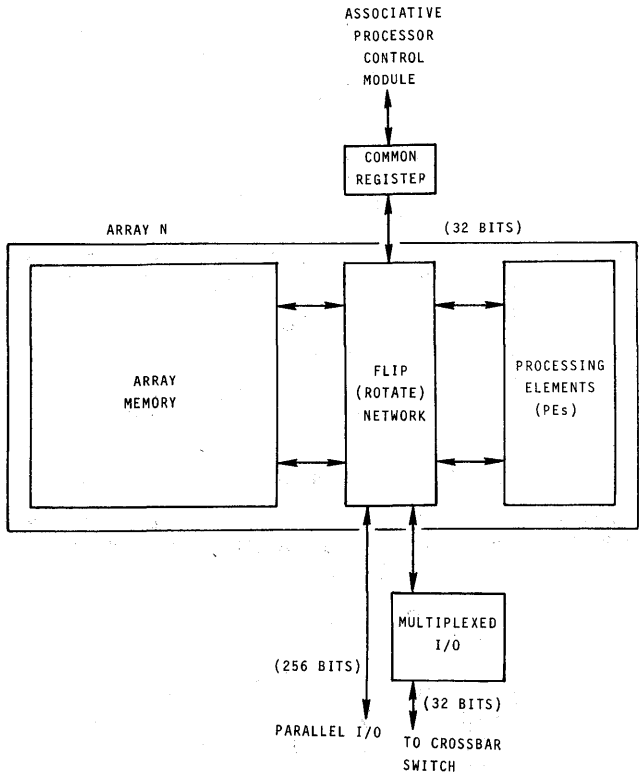


Figure 9—Array I/O paths

required. Consequently, it is best suited for special purpose applications.

Every array module is capable of being connected to either of two controllers. Thus in those applications where the equivalent of double buffering is desirable, some of the arrays can be switched to an I/O controller while the remainder are used for computation.

SUMMARY

The multiple data stream parallelism of the STARAN allows it to perform a "serial" algorithm on up to 8192 data elements simultaneously. This is important in image processing where large volumes of data are processed. Images are inherently two dimensional, but the large array size of the STARAN E, readily allows an entire 512×512 8 bit/pixel image to be stored in one array with essentially its two dimensional topography intact. The array addressing structure and the array flip network provides easy access to every pixel and its neighbors in a simple efficient manner. The mask (M) register operation in an array enables complex decision processes to be made on any subset of pixels in an array. All of the above aspects of the STARAN's architecture would not be valuable if it could not be efficiently used. The three I/O paths into every array provide the capability to efficiently broadcast parameters and constants as well as loading and unloading image data in an expeditious manner. Thus it is apparent that many of the architectural features of STARAN are ideally united for image processing and pattern recognition. This conclusion is confirmed by the execution times shown in Table I.

TABLE 1.—Sample Algorithm Processing Times

Function	Description	Image Size	Speed*
Magnification	2.5× cubic convolution interpolation	512×512 8 bit/pixel	588 milliseconds
Convolution	3×3 window	512×512 8 bit/pixel	700 milliseconds
FFT	1 Dimension	512 16 bit/pixel	2.7 milliseconds

* Measured times in the STARAN B machine exclusive of I/O. The STARAN E instruction execution time is approximately 20 percent faster.

REFERENCES

1. Batcher, K. E., "STARAN Parallel Processor System Hardware," *1974 Computer Conference,* AFIPS Conf. Proc., Vol. 43, pp. 405-410.
2. Batcher, K. E., "STARAN Series E," *1977 International Conference on Parallel Processing.*
3. Goodyear Aerospace Corporation, "The STARAN E System—An Overview," GAC Document Number AP-123226, 29 September 1976.
4. Goodyear Aerospace Corporation, "Application of STARAN to Fast Fourier Transform," GAC Document Number GER-16109, 31 May 1974.
5. Rohrbacher, D. and J. L. Potter, "Image Processing with the STARAN Parallel Computer," *Computer,* Vol. 10, No. 8, August, 1977, pp. 54-59.
6. Gambino, L. A. and B. L. Schrock, "An Experimental Digital Interactive Facility," *Computer,* Vol. 10, No. 8, August, 1977, pp. 22-28.

The criterion COBOL system

by MICHAEL D. SHAPIRO
NCR Corporation
San Diego, California

INTRODUCTION

Higher level programming languages provide problem solvers an access to computers without requiring them to become computer experts. In the traditional environment, compilers perform the relatively complex translation from the higher level language statements into the machine instructions. The differences between the complex data structures and operations of languages and typical simple structures of machines makes this a wide chasm to bridge.

The mismatch between data and operations of the languages and machines appears clearly when one considers a language such as Cobol, APL, PL/I or Snobol4. In each of these, the fundamental operations and even basic data types do not translate easily because the language design depends on no one machine. The implementor must build mechanisms ranging from full software interpreters to elaborate subroutine libraries to perform the higher level language operations on higher level language data.

Despite the costs, the payoffs of using higher level languages make them worthwhile. And with the widespread use of microprogrammable processors as the base of computing systems, designers can realize the once primarily theoretical higher level language machines.[1] An interpreter for the operations of a higher level language, written in firmware rather than software becomes a higher level language machine.

The NCR Criterion Cobol System implements an indirect higher level language machine for 1974 ANSI Cobol in a conventional microprogrammed environment, permitting execution of a single run unit composed of both Cobol and other language modules in a highly efficient manner.

The system consists of a microprogrammed hardware processor supporting both conventional and Cobol machine firmware in a multiple virtual machine environment, a multiprogramming virtual resources operating system, a 1974 ANSI Cobol compiler designed to run in a virtual storage environment, the necessary execution time support subsystems and routines, and Cobol symbolic debug and runtime tuning subsystems.

The discussion focuses on the characteristics of the system as a Cobol environment and the structure of the Cobol virtual machine, the compiler, and the support subsystems.

THE CRITERION SYSTEM

NCR system architects chose a higher level language approach as part of the basic design of the Criterion, a new family of computers to replace the aging Century line. The other part of the design fully supports, without modification, existing NCR Century application programs, no matter what their programming language.

Cobol, probably the most widely used data processing applications language and hence of most interest to the company, serves as the base of the higher level language system. The new virtual resources executive (VRX) operating system design includes support of multiple virtual machines. Cobol Virtual Machine (CVM) firmware, a new American National Standard Cobol[2] compiler, and a set of Cobol execution time support subsystems form the resulting Criterion Cobol System (CCS).

A very fast (56 nsec) pipelined microprogrammable processing element forms the base hardware of the Criterion system. Three storage units attach to it: register storage (RSU), instruction storage (ISU) and memory storage (MSU). The RSU holds temporary work areas for the element. Microcode executes primarily out of the ISU, although some low-speed instructions execute from the MSU. The MSU areas not used by firmware become the main storage used by the operating system and application programs.

The hardware supports a paged memory environment with a maximum addressing capability of 16 megabytes (MB). The processing element performs address translations with interrupts to the executive on page faults.

The customer can receive either of two sets of firmware and software for the Criterion: the real resources system (RS1) or the virtual resources system (VS1). RS1 emulates NCR Century computers and permits use of the Criterion as a direct replacement. VS1 supports the virtual resources, including virtual storage and the multiple virtual machines. The two current virtual machines provide an extended version of the Century instruction set and architecture, supporting the NEATVS assembly language (NVM) and the new CVM.

While these two machines co-reside in the system and support many similar elementary data types, they have sev-

eral fundamental differences in design orientation. NVM uses binary data extensively while CVM works best with decimal data. NVM uses an absolute addressing scheme, but with addressing a maximum of 32 kilobytes (KB) offset from a register, except in the first 64 KB. CVM uses bases, but allows offsets up the full virtual address space limit.

Both NVM and CVM share a common stack-oriented calling mechanism and a common virtual addressing scheme so that a program executing in either machine mode can invoke a module in the other. Each entry point to a module has an entry point control item (EPCI) indicating the virtual machine mode for execution of that module. Machine state can change on any call.

COBOL VIRTUAL MACHINE

The Cobol Virtual Machine (CVM) implements an indirect higher level language machine for the Cobol data types and structures and for a large subset of the Cobol operations. An object module generated for execution by the CVM contains three principal memory areas and some minor auxiliaries.

The data area contains all the data described in the program either explicitly or implicitly. Implicit data includes various file tables, collating sequence tables, debugging tables, and literals. Data definition entries in the Data Division describe explicit data.

The descriptor area contains an internal representation of the information contained in the data description entries. Each referenced data element has a descriptor. Each descriptor contains location, size, type, index, and editing information for the data element.

The code area contains the translated procedure division statements. The basic internal operations (for example, simple MOVE or ADD) translate to single machine instructions. More complicated statements, such as those having multiple receiving fields, decompose into simple Cobol statements for code generation. With operation codes generally typeless, automatic conversion and editing occur when required by the referenced descriptors.

Data types

The fundamental elementary data types in the CVM correspond to the variations on COMPUTATIONAL and DISPLAY types in Cobol.

Packed decimal items represent computational data internally. The requirement for frequent conversions makes packed decimal the most efficient format to use in CVM. Depending on the Cobol picture, the item may or may not have a sign. Computational data items may vary in precision from 1 to 18 digits and may have an implied decimal point at any position adjacent to a digit or may have scaling to a maximum of 18 positions from the farthest digit. An additional four-bit trailing half-byte represents the sign, when present.

Both numeric and nonnumeric data items occur. Numeric display items follow the same rules for computational items on precision and decimal point position. ASCII characters '0' through '9' represent the digits. If the programmer specifies a signed item with a "separate sign," an ASCII '+' or '−' represents the sign. The user specifies whether the sign goes before or after the digits. If the user requests a "zone sign," the sign indicator combines with the first or last digit to form an ASCII character according to this table:

no sign	0	1	2	3	4	5	6	7	8	9
+ sign	{	A	B	C	D	E	F	G	H	I
− sign	}	J	K	L	M	N	O	P	Q	R

Strings of ASCII characters form the nonnumeric data items, having the internal categories of alphanumeric, alphanumeric justified right, numeric edited, and alphanumeric edited, and the properties specified in the Cobol standard.

The CVM provides two additional data types. Signed binary (two, four, or eight bytes) provides data communications with the NVM. User documentation recommends against its use for other purposes. Special index-handling commands manipulate index data items for the table handling operations. The internal structure of these items remains transparent to the user.

Descriptors

The descriptor area may contain any combinations of four types of descriptors. Two of the descriptors, used for compact data descriptions, require four bytes each. Only certain elementary items use them. The other two descriptor types define the more complicated data structures. The long descriptors take eight bytes each.

Short nonnumeric descriptors describe alphanumeric or alphanumeric justified right data items whose length does not exceed 4095 characters. Short numeric descriptors describe decimal, packed decimal, or binary items. In both cases, the item must begin in the first 32768 bytes of the data area.

Long nonnumeric descriptors describe alphanumeric, alphanumeric justified right, or alphanumeric edited data items. They allow a maximum length of 65535 characters. Other fields describe the number of dimensions the item has, whether an edit information mask pointer follows, and the method of addressing used for the data. Long numeric descriptors vary from the long nonnumeric descriptors in that they have decimal position and length fields in place of character length fields.

Edited data items require edit information. An edit mask provides the description of the format of the result. The mask derives from the picture specified in the source program. For alphanumeric edited and numeric edited items, a flag in the descriptor indicates an extension to twelve bytes, with the additional four bytes pointing to the edit mask and containing the numeric length for numeric edited items.

Addressing

Addressing in the CVM uses the standard 24-bit Criterion virtual address mode. Within a program, the system com-

putes the effective virtual address as an offset from the start of the data, descriptor, or code area. A linkage table contains the three bases. On entry to a module, the XLINK register of the machine points to a table of bases. The first three entries contain bases for data, code, and descriptor areas. An additional eight entries point to various other control and debugging locations.

Effective data addresses for descriptored items come as direct offsets for the data area or compute from addresses passed through parameters. Automatic computation of indices for tables occurs when table references appear.

Code

The CVM instructions, as indicated earlier, closely match Cobol statements for their operations. The common statement options form part of the instructions, where appropriate: numeric operations have rounded and size-error flags.

Each operation contains a bit indicating whether it begins a paragraph. The debugging system, described later, uses this information.

A few of the typical commands demonstrate the nature of the CVM instruction set:

MOVE commands act like simple move statements in Cobol, reformatting data as necessary from the source to the destination. A single instruction provides for all conversions and editing. The MOVE-BYTE commands allow movement of a single character literal.

COMPARE commands form the basic operations of the IF statements. The principal COMPARE command handles both numeric and nonnumeric data. Separate instructions provide IF-ALPHABETIC and IF-NUMERIC tests. Comparison tests use either native ASCII collating sequence or one derived from some table, such as EBCDIC. The user specifies the choice with a PROGRAM COLLATING SEQUENCE clause.

GOTO commands provide for internal branching, with Cobol GO TO and GO TO . . . DEPENDING statements as basic instructions. Others include a conditional GO TO for size error detection and an indirect GO TO for PERFORM handling.

Arithmetic commands mimic the Cobol arithmetic statements (except the COMPUTE statement). The commands require numeric source operands and numeric or numeric edited result items. Both two- and three-address operations occur, corresponding to statements of both forms:

```
ADD A TO B.
ADD A, B GIVING C.
```

Programmers request rounding and size error detection in the arithmetic commands by clauses in source:

```
ADD A, B GIVING C ROUNDED;
ON SIZE ERROR . . .
```

CALL commands invoke a standard Criterion calling sequence. The program sets up parameters in the standard parameter stack using a MOVE-VMA command to compute the virtual address. A state switch to NVM may occur during the call.

EXIT commands provide the Cobol EXIT PROGRAM statement for a called subprogram. On exit to the calling program, a state switch may occur to return to the NVM state if an NVM program invoked the CVM program.

THE COBOL COMPILER

The Cobol compiler (CBL) for the CCS converts 1974 ANSI Cobol programs to CVM object modules. Its logic design generally matches that of a typical textbook example Cobol compiler, but with the unique feature of its extensive use of the Criterion virtual address space.

The implementation team used contemporary software engineering techniques in constructing the compiler. They wrote the bulk in a subset of the NCR Software Writer's Language (SWL). NEATVS assembly language functions provide a few utility services, and Cobol coded modules (after bootstrapping) do the input-output. CBL contains more than 200 modules, linked together in a tree-structured manner.

The development process, which occurred concurrently with the development of the Criterion and the operating system, heavily used a variety of tools. A commercial time-sharing system served as the initial development engine. An earlier Cobol development project formed a base for about 30 percent of the code. Snobol4 encoded programs translated this code to the SWL syntax required by CBL. The compiler team coded the remainder of the compiler in the SWL subset. A tools group developed a compiler for the time-sharing system to match the subset of SWL used by CBL. The resulting object program for CBL executed under a simulated virtual storage test environment developed for the CBL project. The team debugged most of the compiler logic before the new hardware and operating system became available.

When the Criterion system could run with a rudimentary operating system, a modified remote entry system transmitted source modules to the NCR equipment for compilation and testing. This linkage remained in place throughout the development cycle. The master library remained on the time-sharing system until the Criterion system became fully stable.

User's view of CBL

To the Cobol programmer, CBL appears as an easy-to-use conventional compiler. Information provided at compilation time relates the execution time debugging output to the source program. The compiler can produce extensive diagnostics in clear English. The programmer specifies a

requested minimum level, from non-ANSI feature through serious errors only. Diagnostics refer to the source program by source line and column number.

Input to the compiler comes from source cards, from a Cobol source library, or (for compatibility with earlier systems) from NCR "SPUR" format files. Outputs include the source listing and a diagnostic summary, along with the object module on disc, if the compiler generates it. The user may specify optional cross-reference, object listing, and object map printed output. The Cobol library facility furnishes texts for the Cobol COPY statement.

Object modules produced by the compiler contain the code, data, and descriptor tables required by the CVM, the control tables used by the link editor, and optionally, debugging tables used by the execution time Cobol symbolic debugging and tuning systems.

Tables and texts

The structure of internal tables and texts gives CBL the properties which make it particularly well-suited for virtual storage operation. As used in this discussion, a table structure holds randomly accessed data and a text structure stores sequentially accessed data.

Tables in CBL contain the information usually associated with the compilation process: data names and attributes, procedure (label) names and attributes, file properties, literals, etc. In keeping with the rules for virtual storage programming, the design of tables makes them compact. A vector of data records forms a table. If a record requires eight or fewer bytes, the table consists of the collection of records. If the record size exceeds eight bytes, the table contains a vector of three-byte record pointers and a collection of records allocated, as required, from a storage pool. This approach permits CBL to compile the widest mix of Cobol programs, keeping the table storage as compact as possible.

A few principal internal tables hold most of the data passed between phases:

DATA TABLE. The data table contains the data name information and the attributes of the data name. The attributes include both external properties, such as line and column where defined, and internal properties, such as data type or level in structure. An auxiliary table entry holds less frequently used items which cannot fit into the data name attribute table.

PROCEDURE TABLE. The procedure table contains the procedure name (Cobol paragraph and section label) information. It uses the auxiliary table when necessary.

PICTURE AND LITERAL TABLE. The picture and literal table contains pictures and literals which appear in the source program, literal generated by the compiler, and edit masks generated from the pictures for numeric edited and alphanumeric edited data items.

FILE TABLE. The file table takes the information developed by the compiler from environment, data, and procedure division references to files. The file table links to the data table and to the auxiliary table to store additional information.

Texts in CBL store the information frequently associated with temporary scratch files. A text appears as a large vector of relatively small records. The compiler places entries into a text or retrieves entries from a text in a sequential manner. In this way the virtual storage paging mechanism moves the idle information to the page storage device and brings it back when referenced. The overhead associated with normal file input-output operations disappears and, for a small Cobol program in a relatively idle system, the text pages may never need paging out.

The principal internal texts used for passing information between phases in CBL are:

COBOL TEXT. The Cobol text (CTEXT) contains a regularized representation of the original source program with data names, procedure names, and reserved words, pictures, and literals represented by entry numbers in appropriate tables.

GENERATOR TEXT. The generator text (GTEXT) contains an intermediate form of the Cobol program, with references to table entries for data items, labels, literals and other items. The GTEXT contains a series of "packets" which correspond roughly to assembly language statements in a compiler which compiles to assembly language in the "front end."

OBJECT TEXT. The object text (OTEXT) contains the object information produced by the object generator, but before packaging into an external object module format.

ERROR TEXT. The error text (ETEXT) contains the information produced by the compiler analysis phases and used by the diagnostic formatter.

Compiler phases

CBL divides into from five to ten phases, depending on how one wishes to number them. If one chooses the minimum number, several of the phases divide into subphases.

A main driver initializes compiler work areas, opens source and listing files, processes user request options and then calls the first compiler phase. It proceeds through each compilation phase, checking for a possible compilation abandonment after each, until it reaches the object generation phase. It invokes generation only if:

a. the user has not suppressed it (an option), and
b. no serious error has suppressed it.

After object generation, the object formatter outputs the code, the main driver prints a compilation summary, closes

the files, and returns to the monitor to terminate compilation.

The source scanner, first phase of the compiler, converts the Cobol source lines read from input to the CTEXT. A hashing mechanism (buckets and chained entries) builds the symbol table through the data and procedure tables. Pictures, literals, and unknown symbols go into the picture and literal table. The scanner produces the source listing and places syntax diagnostics into the ETEXT.

The source scanner does all the Identification Division processing, principally recording the program name.

Three subphases handle the Environment and Data Division processing. One of these, the picture analyzer, scans the picture strings for meaning and errors, and sets up attributes for data associated with the picture. It also builds edit masks corresponding to specifications in the picture. Two subphases extract source from the CTEXT and parse and analyze other table information in the divisions, building and modifying the appropriate table entries. All three subphases can generate diagnostic requests into the ETEXT.

The Procedure Division processor takes the source text from the CTEXT and information from the tables and builds the GTEXT. It also makes entries, if necessary, in the ETEXT and other tables.

The final front-end phase, the literal pooler, moves entries from the picture and literal table into the literal pool, a small text.

Two back-end subphases generate the object module. An object generator analyzes the tables and literal pool to build the data and descriptor areas. It uses the results from these and the GTEXT to construct the code area, and also produces the required link editor tables. All of this information goes into the OTEXT, which passes to the object formatter. The object formatter writes the object module on disc and optionally produces an object map and listing.

The main program has two utility phases available when it needs them. A cross-referencer produces an alphabetic reference listing of all data and procedure names. The main program runs this procedure, on user request, immediately after the source scanner. The diagnostic formatter runs just before the end of compilation if any phase has emitted a diagnostic request to the ETEXT.

EXECUTION TIME SUPPORT

Criterion Cobol System execution time support includes a set of "global" routines for handling the operations required for input-output, sort-merge, communications, and operating system interface statements. It also contains software/firmware debugging and tuning subsystems, described in the next section.

Global routines in VRX, residing in a protected memory area accessible to all simultaneously executing programs, consist of reentrant code.

The input-output system, the Criterion Access Method (CAM), provides Cobol support for sequential, relative, and indexed files. Relative and indexed files may reside on any disc type supported on the machine. Other devices handle sequential files only.

The sort-merge system provides the Cobol support for sorting and merging magnetic tape and disc files.

The Message Control System (MCS) provides functions for the Cobol Communications functional module. The Network Description Language (NDL) processor sets up the communications paths according to user specifications.

An operating system interface module, invoked primarily for Cobol ACCEPT and DISPLAY statements, supports such actions as reading the clock or calendar, communicating with the operator, and reading control card parameter information.

COBOL SYMBOLIC DEBUG AND TUNING AIDS

The Cobol symbolic debug system (COBUG) provides the Cobol programmer with a variety of information for symbolically monitoring the program and data flow in an executing program.

The first instruction of each paragraph contains a special flag bit. Before executing the instruction, the firmware checks that bit and an internal symbolic debug switch. With both conditions *on*, the COBUG subsystem takes control. It displays information requested by the user. The user, through control statements, may request a variety of types of information:

DUMP of all or selective data items at specific places in the program. Options include specification of which time during execution the dump should occur, the number of times, and relational conditions:

```
DUMP ABC AT XYZ FOR 10 TIMES WHEN A
IS LESS THAN B
```

TRACE of all or selective paragraph flow. Options include ranges, and the number of times:

```
TRACE XYZ TO PDQ ON 5 FOR 12 TIMES
```

FLOW of program execution. If requested, the firmware records the execution of each instruction. At program termination time, the COBUG system formats and prints a listing of the data. The user, who must understand the CVM operation, can use this listing to detect problem points in the program.

The Cobol analysis routine for runtime tuning (CARTUNE) provides the Cobol programmer with a mechanism for collecting execution time statistics. The Cobol compiler, on request, builds a source line table and three activity tables in the object module. At execution time, the CVM firmware optionally maintains tables of the number of times each type of operation code executes, the number of references to each data item, and the number of times each instruction executes.

At the end of execution, CARTUNE gathers the information and prints an analysis. The user may request specific

reports: source line executions, data references, and object instruction execution history. These data assist the programmer in locating the inefficient portions of the program for improvement.

THE RESULT

The Criterion Cobol System provides the Cobol user with an integrated system optimized for highly efficient Cobol program development and execution.

ACKNOWLEDGMENTS

CVM designers and implementers:
Ward F. Hardman, Jr.
C. Kent Huckstep
Donna G. Hynes
Donald G. McCrimmon
John D. Roberts
William F. Schweitzer
Michael D. Shapiro
Robert A. Surtees
Wayne H. Uejio
William W. Woo
John W. Young, Jr.
Cobol compiler team:
William A. Aebi
T-C Chao
Charles L. Citron
Dana M. Foy
Donna G. Hynes
Donald G. McCrimmon
Neal F. Openshaw
Ray L. Paul
Don A. Schricker
Michael D. Shapiro
Dale A. Shoemaker
Robert A. Surtees
Cobol symbolic debug and tuning aids:
Dana M. Foy
Larry R. Luko

REFERENCES

1. Chu, Yaohan (editor), *High-level Language Computer Architecture,* New York, Academic Press, 1975.
2. American National Standards Institute, *American National Standard Programming Language Cobol,* ANSI X3.23-1974.

Review of the CLIP image processing system

by M. J. B. DUFF
University College London
London, England

INTRODUCTION

In this paper, the terms "pattern recognition" and "image processing" will be considered only in relationship to optical or visual images, which will be expressed as two dimensional arrays of numbers, each number representing the image intensity in the corresponding picture element, or "pixel." The image will be assumed to be adequately reproduced by means of an n by n array of square picture elements whose intensities are chosen from a range of L discrete values which are uniformly spaced over a selected brightness range. Obviously, both n and L must be large enough to give acceptable spatial and grey-tone resolutions for the task to be performed on the image.

Many of the operations carried out during pattern recognition or image processing can be described as "local;" this implies that the new value of a pixel in the processed image is a function of the values of a limited and compact subset of pixels in the original image. A particularly useful subset is the immediate neighbourhood of each pixel, i.e. the three by three array of adjacent pixels surrounding and including each pixel. The implication of this statement is that extremely fast processing of images could be achieved by constructing an array of identical processors in one to one relationship with all the pixels of an image, each processor receiving inputs from its corresponding pixel and from the eight immediate neighbours of the pixel. The purpose of the CLIP (Cellular Logic Image Processor) programme of research has been to optimize a processor array structure for the range of operations commonly required in image processing, having due regard to the cost of implementing the system as an array of large scale integrated circuits. At the same time, it was recognized that, ideally, the array should be capable of executing all possible functions provided an appropriate sequence of allowed instructions is obeyed. In other words, the array should perform as a general purpose computer optimized for a typical range of image processing operations.

CLIP4 LOGIC

The basic unit of the array is the cell, being a combination of logic units (i.e., the processor) and memory. Details of the CLIP4 cell are shown in Figure 1(a) in which A, B and C are single bits of buffer memory and D is a 32 by 1 bit RAM. The boolean processor provides two program selected independent functions, D and N, of the two binary inputs A and P, one of which (N) is used to form an output to all neighbouring cells. Inputs from neighbouring cells are individually gated and then OR'ed together to form the binary function T which combines with B to produce P. A and B are each single bits of the pixels of two images in the array location corresponding with the cell concerned. Other gates are included which can be used to generate the exclusive OR (EXOR) of B and T at P and also the AND of B and T which is then OR'ed with the N output of the boolean processor to form the interconnection output N*. These additional gates are of use when the processor is performing arithmetic operations and are brought into use by special instructions which put 1's on the R and C control inputs.

Before considering the details of the operations in the CLIP4 cell, it will be helpful to examine the organisation of the data storage in the n by n cell array (see Figure 1(b)). Each address in the D memories can be visualised as a plane of single bit stores, d. If the location of each cell is given by its (x,y) coordinate in the array, then a bit plane will be the set of bit stores defined by:

$$D_j=\{d_{jxy}: x,y=1, 2, \ldots n\}$$

A pixel will be formed as a column of single bit stores passing through g bit planes. Thus the pixel at location (x,y) is defined by:

$$P_{xy}=\{d_{jxy}: j=k, k+1, \ldots k+g-1\}$$

D_k will contain all the least significant bits in the binary numbers representing the grey levels in each pixel; D_{k+g-1} will contain the most significant bits. Note also that

$$g=\log_2 L$$

implying g binary bit planes are needed to represent L grey levels. The ordering and addressing of the bit planes in the 32 available addresses in the D memories is arbitrary and may be varied to suit the needs of the program.

When numerical calculations are to be performed on data in which there is no longer an exact correspondence between the n^2 pixels and the n^2 processors, it is sometimes convenient to represent data in the binary column mode, in which n binary numbers are stored in the n columns of a

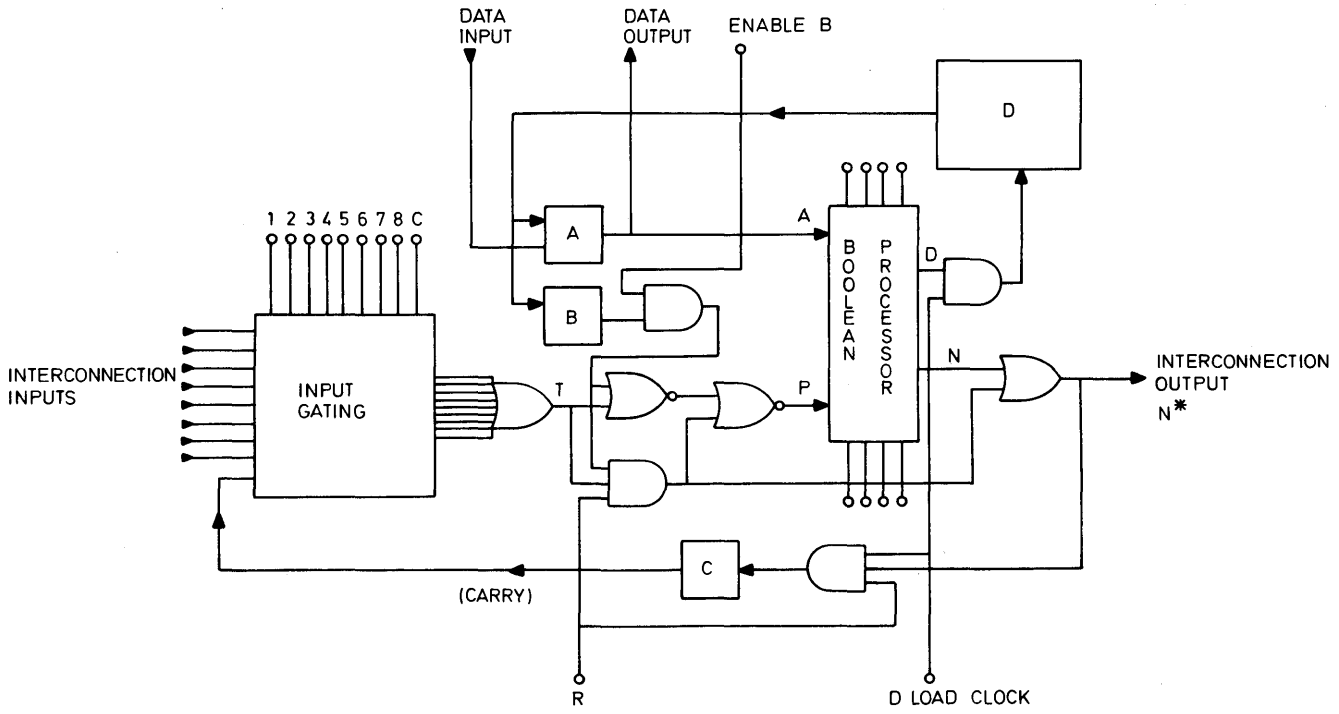


Figure 1(a)—The CLIP4 logic and memory structure

single bit plane. A binary number will be represented by the column:

$$\{d_{jxy}: y=1, 2, \ldots, n\}$$

Normally, the precision provided by n bits will be unnecessarily high for typical values of n. Methods have been devised for using the available storage more effectively.

A binary image is composed of elements which are either black or white so that pixels have values 0 or 1 respectively. Usually, binary images are produced by thresholding a grey tone image. Binary images are stored in a single bit plane of the D memories, at any convenient address.

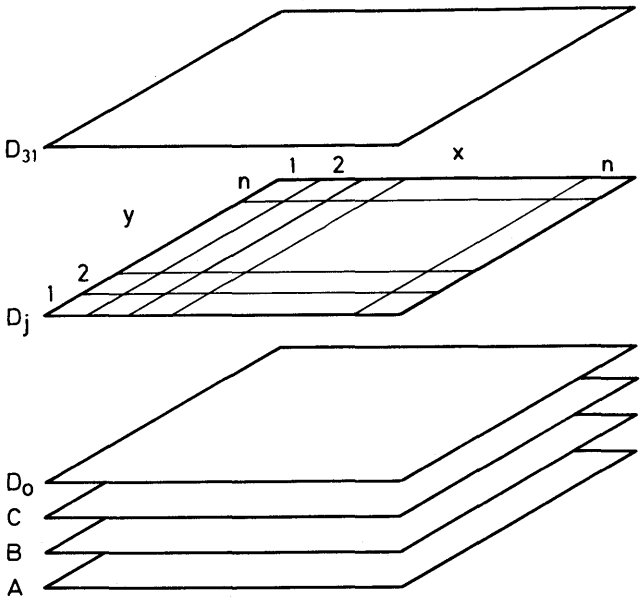


Figure 1(b)—Bit plane and binary column data organisation

USING THE CLIP4 CELL

The five principal ways in which the CLIP4 cell can be configured are illustrated in the simplified logic diagrams of Figure 2(a)-(e). In each case, the symbols A, B, C and D refer to bit planes.

Simple boolean operations (Figure 2(a))

Binary image I_1 is loaded into A and binary image I_2 loaded into B. The direction gates 1 to 8 and inputs R and C are disenabled. Any of the 16 possible boolean functions of the two images can be produced and loaded into a selected address of the D memories. If I_1 is in D_1, and I_2 in D_2, and if the result address is D_3, the coding for the operation of taking the exclusive OR of I_1 and I_2 is (in the CLIP4 language: CAP4):

```
SET  P @ A
LDA  1
LDB  2
PST  3
```

(Notes: @ implies EXOR; PST is a mnemonic for Process and Store)

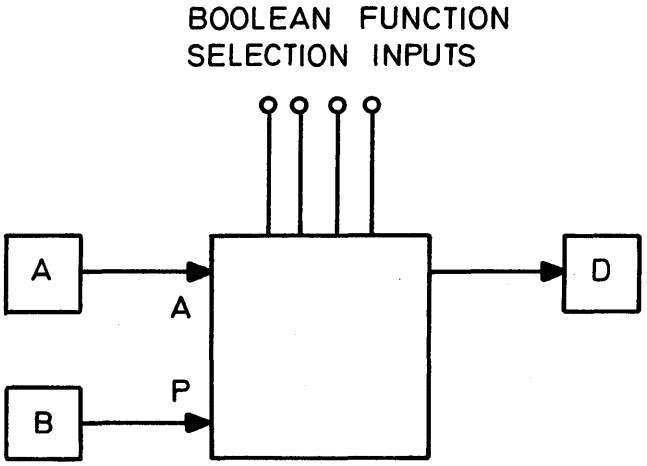


Figure 2(a)—Simplified logic for boolean operations

Simple propagating operations (Figure 2(b))

A binary image is loaded into A and an interconnection function set up (for example: N=A). A selection of interconnection gates are opened (for example: those in directions such that outputs from neighbouring cells to the North, East, South and West are admitted, i.e. directions 2, 4, 6 and 8) and a boolean function for the output D is chosen (say: P=$\bar{A}$). The coding is:

```
SET  P + − A, [2 4 6 8] A
LDA  1
PST  3
```

(Notes: the negative sign implies NOT; the B pattern does not affect the operation).

Labelled propagating operations (Figure 2(c))

In this case propagation can be started from any cell in which B has the value 1, i.e., from any white pixel in I_2. Using the same values as in the previous example, the coding becomes

```
SET  P+−A, [2468B] A
LDA  1
LDB  2
PST  3
```

(Notes: the connection to the B store is enabled by inserting the symbol B at the end of the direction list on the SET instruction).

Propagation can also be initiated at the edges of the array by adding an E at the end of the SET instruction. This has the effect of supplying a 1 to all interconnection inputs of array edge cells which cannot connect to missing neighbours lying outside the limits of the array.

Bit plane arithmetic operations (Figure 2(d))

If A is a particular digit of one binary number, B the corresponding digit of another binary number, and C the carry digit resulting from the addition of the previous digits of less significance, then the expressions for the sum digit and new carry digit are:

```
SUM=A.EXOR.B.EXOR.C
CARRY=(B.EXOR.C).OR.B.AND.C
```

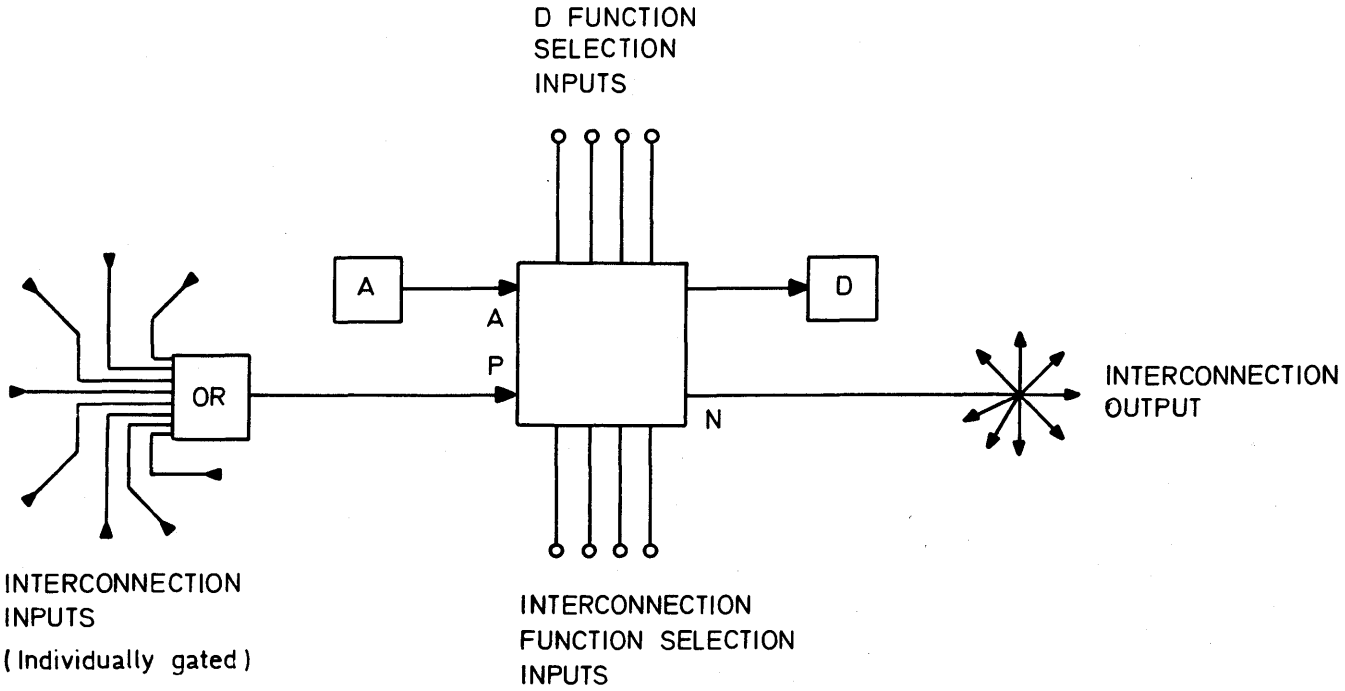


Figure 2(b)—Simplified logic for propagating operations

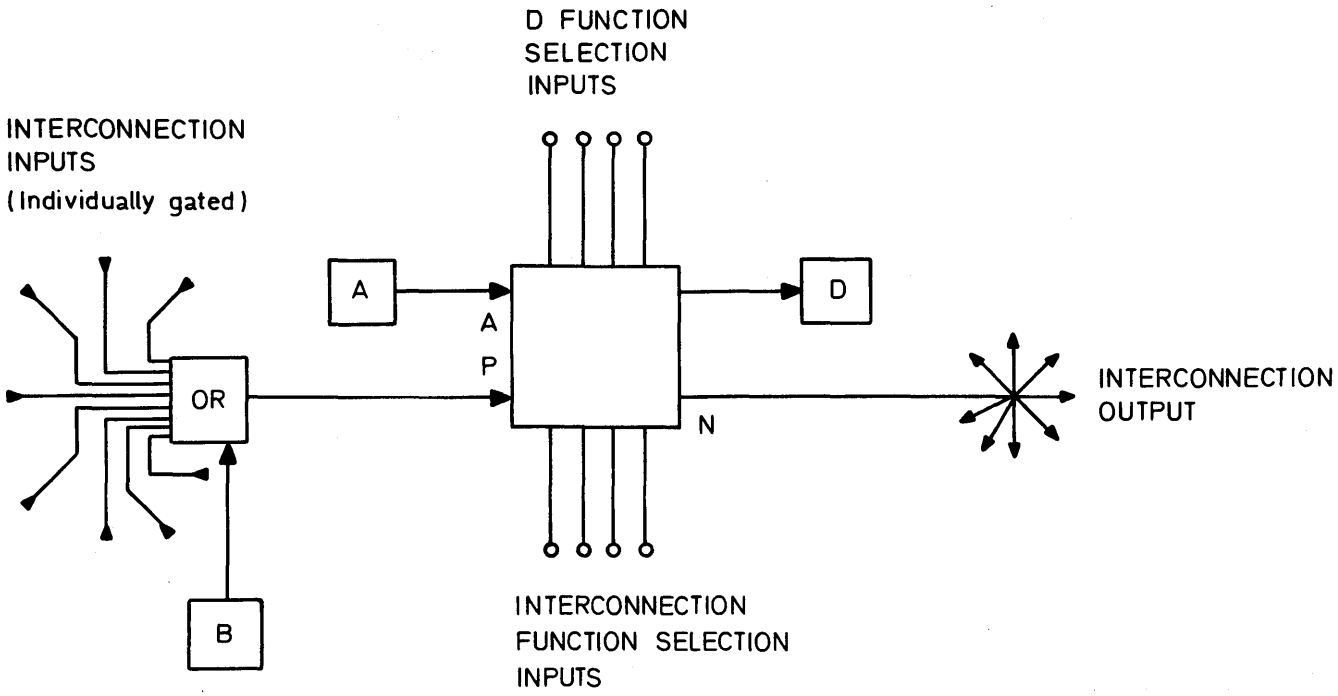


Figure 2(c)—Labelled propagation logic

When the inputs R and C are enabled, the CLIP4 cell will implement this arithmetic function for bit planes loaded into A and B, using the boolean processor to provide the operations N=P.A and D=P.EXOR.A. If two grey tone images are loaded into D_1 to D_6 and D_7 to D_{12} respectively, then the part of the coding which adds the second least significant digits is:

```
SET P@A, [B] P.A, RC
LDA 2
LDB 8
PST 14
```

Giving a summed image in D_{13} to D_{19}.

Binary column arithmetic operations (Figure 2(e))

Two planes of binary column numbers in D_1 and D_2 can be summed to form a new plane of binary column numbers in D_3 by writing the following short program:

```
SET P@A, [6B] P.A,R
LDA 1
LDB 2
PST 3
```

(Notes: the carry function is automatic when R is enabled; no use is made of the C plane stores; direction gate 6 is opened to allow carries to be propagated along the binary columns).

Other arithmetic functions can be developed from the summing operation in the usual way. In principle, floating point arithmetic can also be performed, although full details of the algorithms involved have still to be worked out.

OTHER HARDWARE FEATURES

In the CLIP4 system now being constructed, a 96 by 96 cell array is being assembled using custom-designed NMOS integrated circuits. Each circuit comprises eight cells arranged in two rows of four. The A elements are serially connected to form 96 shift registers, each of length 96 and lying along the array rows. Single bit planes of data are loaded serially into a 9216 bit shift register external to the array; this shift register can then be reconfigured into 96 short shift registers which then discharge their data into the A rows. The whole system interfaces to a television camera and monitor via two shift register memories, both of which can store a 96 by 96, six bit grey tone image. Analogue to digital conversion and digital to analogue conversion between these memories and the television camera and monitor respectively proceeds at the full video rate.

The array is operated by a controller which extracts the 16 or 32 bit instruction words from the controller memory, decodes them and drives the array and peripherals. Included in the controller are 14 general purpose 16-bit registers. A fifteenth register has a special additional function: a bit plane may be shifted from the array through a tree of parallel adders, thus permitting a rapid count of the 1-bits set in the A stores. This count appears in register 15. Furthermore, another instruction can be used to output the contents of

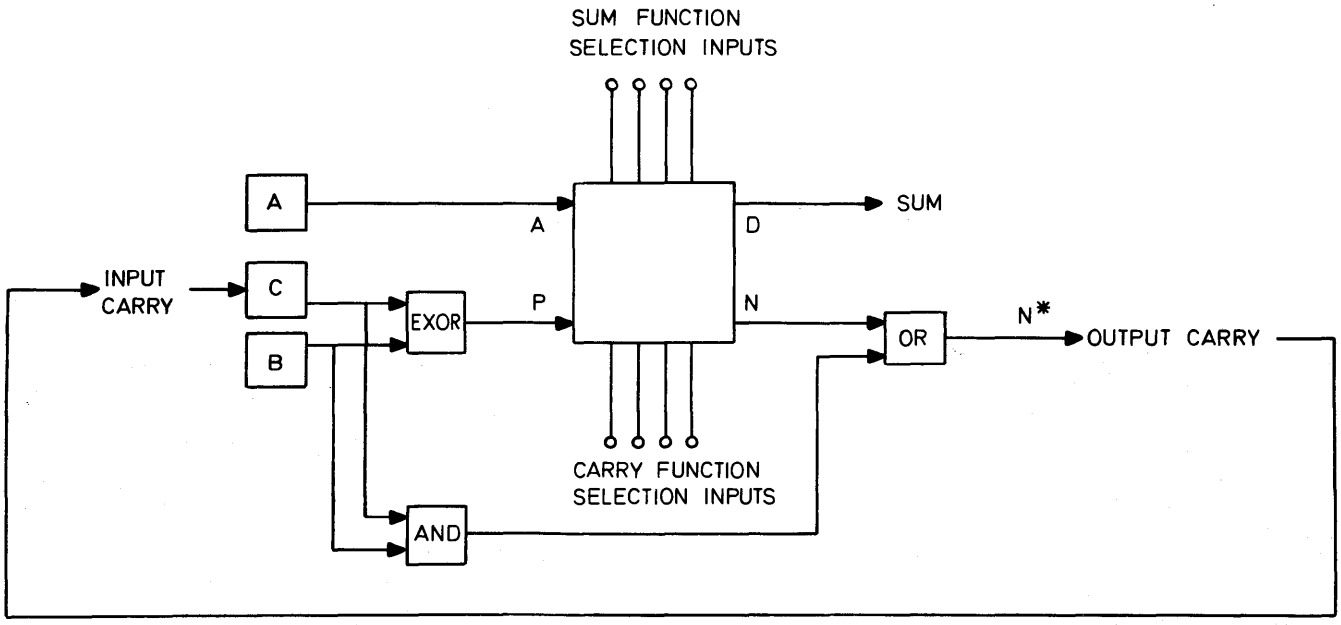


Figure 2(d)—Bit plan arithmetic logic

register 15 back into the array by means of an operation similar to the E instruction.

CLIP4 INSTRUCTION TIMES

CLIP4 is a Single Instruction, Multiple Data stream computer and performs image processing tasks at speeds which are quite unobtainable in conventional machines. The integrated circuit operates with a four phase, 400 ns clock cycle; load instructions require 8 cycles, PST instructions 10 cycles and SET instructions 12 cycles. A further 3 cycles are required for each propagation step through a cell when the selected boolean function is such as to cause propagation. Branch and register instruction require 4 or 8 cycles (the longer time being for memory reference register instructions).

Interleaving of these instructions is used to reduce complete array operations to just under 9 μs plus 1.2 μs/cell for propagation beyond the immediate neighbourhood of each

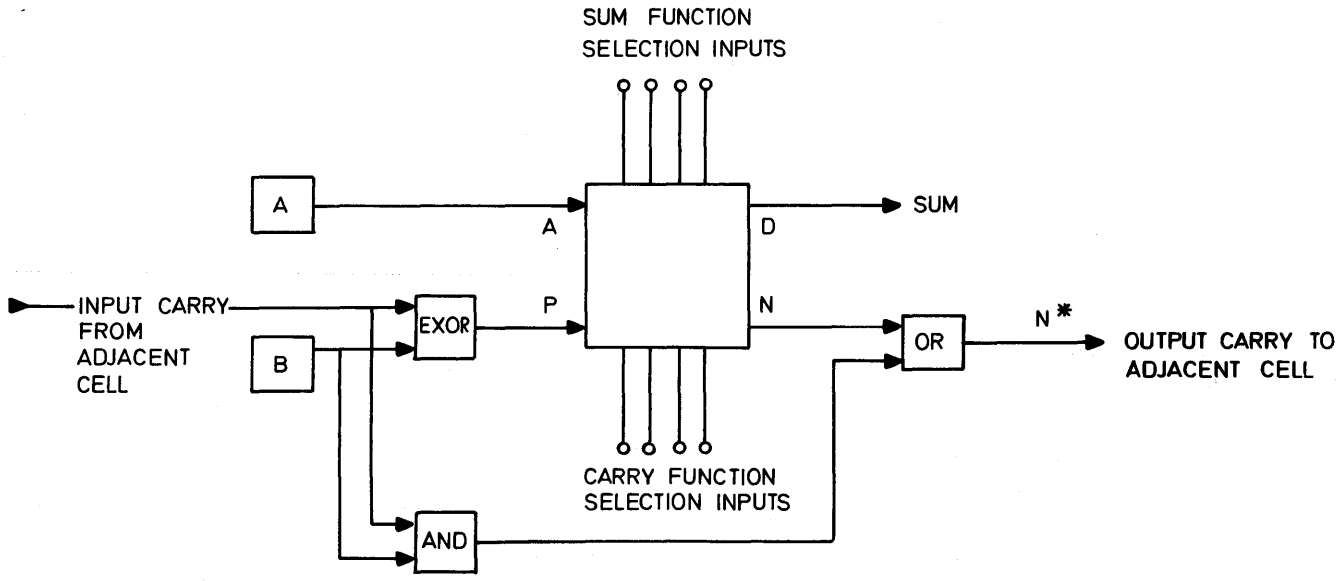


Figure 2(e)—Binary column arithmetic logic

cell. In general, the complete operation will comprise the instructions SET, LDA, LDB and PST. A bit plane of data can be entered into the array in under 4 ms. However, this may increase up to 10 ms since the operation is synchronised with the television scan cycle.

SYSTEM STATUS

A small scale pilot model of the system (CLIP3) has been in operation since 1973. The array comprises 16 by 12 cells and can be used on images of the same size or, by scanning the array across a 96 by 96 pixel image, can be used to simulate the larger CLIP4 array. Additional hardware and software is incorporated to handle sector to sector propagation when CLIP3 is used in the scanned mode. Details of the CLIP3 system are provided in References 1-5, which also include bibliographies of earlier studies both at University College and elsewhere.

At the time of writing (late January 1978), the CLIP4 chip design is complete and prototype chips are expected to be delivered before the end of March 1978. All other parts of the hardware and software for the image processing system are complete, although some modifications and improvements are being carried out. Negotiations are in hand to arrange for CLIP4 systems to be manufactured and made commercially available. Preliminary estimates suggest that the first systems will be ready during mid to late 1979. The University College prototype is scheduled for operation in Autumn 1978.

REFERENCES

1. Duff, M. J. B., D. M. Watson and E. S. Deutsch, "A Parallel Computer for Array Processing," Proc. IFIP Congress 1974, *Information Processing 74*, Stockholm; pp. 94-97.
2. Duff, M. J. B. and D. M. Watson, "The Cellular Logic Array Image Processor," *Comp. Journal 20*, 1977, pp. 68-72.
3. Cordella, L., M. J. B. Duff and S. Levialdi, "Thresholding: A Challenge for Parallel Processing," *Computer Graphics and Image Processing 6*, 1977, pp. 207-220.
4. Duff, M. J. B., "Geometrical Analysis of Image Parts," Proc. NATO Advanced Study Institute, *Digital Image Processing and Analysis*, Bonas, France, 1976, pp. 101-124.
5. Cordella, L., M. J. B. Duff and S. Levialdi, "An analysis of computational cost in image processing," to be published in *Trans. IEEE on Computers*.

Area Director:
Vir A. Dhaka
Xerox Corporation
El Sedundo, California

Evolution of new hardware technology

OVERVIEW

Evolution of new hardware computer technology continues at a rapid pace. Implications of developments in semiconductor LSI technology for logic and memories, new achievements in magnetic bubbles and disk storage and progress in computer printer technology for a modern computer system are far reaching. The impact of these developments has resulted in (i) significant cost reduction for a given function; (ii) increased complexity resulting in enhanced capability and performance; (iii) improved reliability; and, (iv) reduction in physical size. These trends can be expected to continue during the next decade and even though these technological changes are evolutionary, their impact on the computer organizaton and utilization is indeed revolutionary. Entire new areas of application will become available to the computer industry with the advent of inexpensive yet powerful and versatile hardware. The generic impact on the architecture and configuration of a computer will be as follows:

- Every type of terminal, control and data entry equipment, etc., will have a rather large amount of intelligence at the point of use.
- Dedicated small digital systems will be incorporated in computer subsystems, i.e., computer peripherals.
- System design will be determined by repetitive use of very inexpensive hardware.
- Archival mass storage will be available on line.
- Today's operating system features and other system software will be implemented in firmware.
- Performance measurement monitoring, maintenance and error logging accompanied with fail soft and fault tolerant design will be incorporated for increased hardware reliability and availability.

The three key areas of technical progress responsible for the developments projected above are: (1) semiconductor technology for LSI logic, microprocessors and RAMs/ROMs; (2) magnetic technology for bubble memories, and disk storage; and, (3) printer technology for data/information output. Present status and

projected developments in these areas will be described in the sessions on hardware evolution.

Semiconductor technology has been able to double the functional density on a single silicon chip every year since 1960. In the logic area, this progress has taken this technology from a single transistor to IC's and then to microprocessors and microcomputers. In complexity the microprocessors have evolved through 4 bit and 8 bit complexity to 16 bit complexity. By incorporating ever increasing RAM/ROM and control function powerful single chip microcomputers have become available for a price of about $10. Progress in the technology promises that by about the mid-eighties a 32 bit microcomputer with one million bits of memory could be made on a single silicon chip for a cost of about $20. With such inexpensive but powerful intelligence, no equipment would be permitted to stay dumb anymore.

In the area of semiconductor memory, the first challenge to the dominance of magnetic core memories occured in 1970 when the first 1K bit semiconductor memories became available. Since then the progress in this area has been rapid and now the technology stands on the threshold of the availability of 64K bit RAMs. This progression in complexity is likely to continue and reach a 1 million bits per chip by 1985.

CCD and magnetic bubble technologies provide solid state alternatives to rotating electromechanical mass storage devices. During the past year, significant technical advances have taken place in both of these areas. Both of these technologies have started to appear in the commercial products. CCD memories have faster access times than the bubble memories and will find a place in the memory hierarchy. Bubble memories, however, promise to reach the 1 million bit on a chip capability sooner than any semiconductor technology. The resulting low cost and high reliability when coupled with their inherent characteristics of non-volatility will permit this technology to fill the presently existing gap between semiconductor memories and moving magnetic mass storage.

Even as the CCD and bubble memories become more cost effective and attempt to replace magnetic disk technology, the moving head disk technology continues to improve. Disk technology progress has included not only the readwrite heads but media, source coding and error correction and improved system bus interfacing. These developments have improved the performance and reliability. Area density of storage has continuously increased and has been a key factor in achieving cost reductions.

Printer technology developments is another area of major activity. Both the impact and non-impact printing technologies using plain paper have progressed significantly. Ink Jet and xerographic printers are now commercially available. Progress in these technologies is radically changing the quality of the computer output as well as the cost of computer printing.

The technical developments in the areas discussed above will be described and projection of developments in the years to come will be made by experts who themselves have played a major role in making these developments possible.

One more aspect of this rapid evolution in hardware technologies is that wherever there is a significant technological change, there are opportunities for entrepreneur organizations to bring these technologies to the marketplace. A panel of distinguished speakers will address themselves to the proposition of "Opportunities for New High Technology Companies" as a part of the series of sessions on Evolution of New Hardware Technology."

Semiconductor rams of the future

by CHARLES BOETTCHER
National Semiconductor
Santa Clara, California

Semiconductor memories have, since 1970, been a significant factor in data storage. This has been especially true in Random Access memories and is becoming true of the slower serial access memories.

The manufacture of semiconductor memories takes place in two stages; the batch manufacture of the dice as wafers and the piece-by-piece assembly of the good dice into packages. It is the batch manufacturing of the dice that has led to the continuing trend of lower cost per bit. (Figure 1) With each new generation of memories, four factors have contributed to increased density of good bits per batch and therefore lower cost. These four factors are; decrease in storage cell complexity, decrease in feature size, increase in wafer size (thus batch size), and decrease in defect density.

Decrease in storage cell complexity has been the most significant factor in increasing batch density. (Figure 2) An appropriate comparison of cell complexity can be made by expressing cell area in units of f^2, where f is the minimum feature size allowed by the pattern-definition technology. Cell sizes in the last eight years have decreased from $200f^2$ for the first static flip-flops to a present range of $16f^2$ to $20f^2$. It is theoretically possible to reach a cell size of $4f^2$ where there are two features in both the X and Y directions, one to store the information and the other to isolate it from adjacent cells. If means can be found to isolate adjacent cells in less than a feature size, then a cell size of f^2 is possible.

Added to the decrease in cell size is an inherent increase in layout efficiency. When memory size increases by a factor of four, only twice as many decoders, sense amplifiers, etc., are required; only two more address buffer circuits and the same number or less bonding pads are needed.

A combination of design and process innovation has allowed this decrease in storage cell complexity, resulting in a 13.5 times increase in batch density.

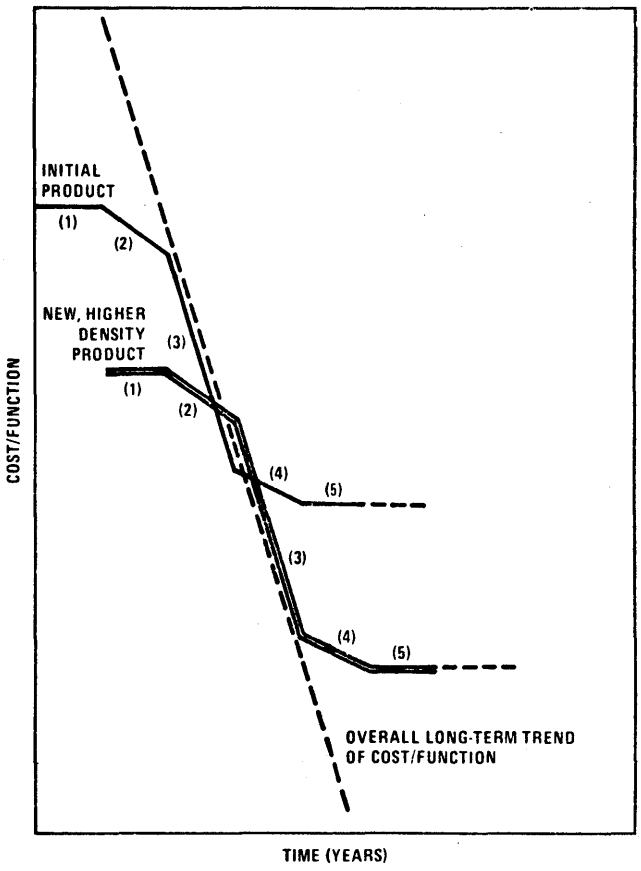
The second most significant of the four factors has been increased wafer size. As a result of a significant amount of development work on the equipment for manufacturing raw wafers and processing wafers, as well as developing processing techniques for handling larger wafers, the size of wafers has increased from two inches to four inches in diameter. This has resulted in a four times increase in wafer area and therefore a four times increase in batch density.

The remaining two factors, feature size and defect density have contributed the least to increasing the number of good bits per batch. Smaller minimum feature sizes have been possible as a result of improvements in mask making, mask aligning, and photoresist technology. Full utilization of improvements has been hampered by several factors. As feature sizes approach the film thicknesses and step heights present, dimensional control and feature integrity becomes increasingly difficult to maintain. In addition, as the feature size is reduced, the size of particle that will cause a defect in the masking is also reduced. Since smaller particles have a much higher probability of passing through the various forms of filtering used in semiconductor manufacturing facilities, the effective density of defects increases exponentially with decreasing feature size. (Figure 3) These factors have limited batch density improvements due to feature size reduction to a 2.2 times increase.

Overall, these four factors have resulted in a 120 times increase in batch density. With each new generation of memories, when a four times larger memory could be fabricated and packaged for the same cost as four smaller memory in four packages, the new generation was introduced. The packaging density increase of course offered an overall savings in system costs and users started switching to the new generation. As volume has increased and yields improved, costs have dropped and the cycle has repeated.

To what extent can we extrapolate and thereby predict the future?

With respect to decreasing cell complexity, there is little real room for progress. Present design and process technology is producing cells in the $16f^2$ to $20f^2$ size range. If we attain a $4f^2$ cell size, we will only realize a factor of four to five times density increase. In addition, in order to attain a $4f^2$ cell size, significantly new device types will have to be developed or adjacent storage cells will have to be built on separate layers. Increasing the number of layers affects yield in the same manner as a larger area. Therefore, achieving a $4f^2$ cell, with a corresponding increase in number of mask layers, may give no actual improvement in good bits per batch. Examples of new device types are Ovonic materials sandwiched between layers at the memory crosspoints or CCD devices which require changes in system architecture to cope with large percentages of their data base in serial access storage. In such cases the technology development required is significant.



NOTES:

(1) Product is introduced, very little or no competition.

(2) Product is multiple sourced but prices still hold up as production is limited.

(3) Production expands dramatically as yields improve, competition is fierce, prices tumble, marginal suppliers are beginning to drop out.

(4) Significant yield improvements are no longer available, new product is beginning to compete for the same market, competition is still intense, suppliers still drop out.

(5) Major market share is taken by new generation of product, market shrinks but prices hold up as competition is minimal.

Data courtesy of T. Klein, National Semiconductor.

Figure 1—Typical decrease of memory products' prices/costs during product life

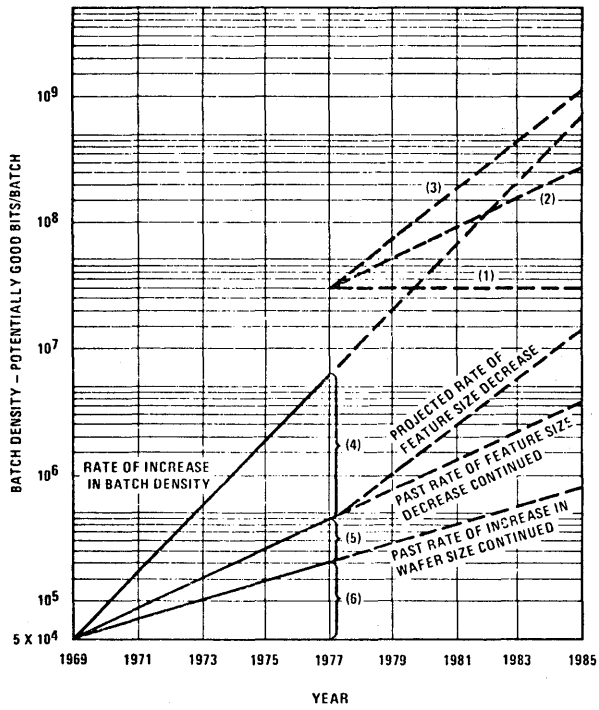
With respect to wafer size increase, the trend will continue, wafer area will double every four years. Each time manufacturers change wafer sizes, virtually all wafer processing equipment has to be replaced at a tremendous cost. Planning ahead and installing equipment that has a longer life is not really possible because of the ability with which equipment can be developed that will handle larger wafers with increasingly stringent control requirements.

Feature size reduction is the area where the industry is predicting the most improvement in batch density.

Projection aligners are already in use or being installed in most manufacturing lines. They have allowed the masks to become permanent tooling rather than production materials that must be discarded with every second or third run, and therefore, made it practical to invest in defect free masks with very tight tolerances. Projection aligners will allow feature sizes approaching one to two microns, a five times improvement over present practice. The lower defect densities possible with projection aligners will help offset the exponentially increasing effective defect density inherent with smaller feature size, but significant and costly improvements will also be needed to reduce other defect sources.

Electron Beam exposure systems are being developed, evaluated and purchased. Proponents of E.B. systems are claiming that they are about to revolutionize the industry. Proper analysis indicates that they will be a useful tool for mask making purposes, but their costs and throughput are such that feature size improvements they make possible will be offset by increased processing costs.

The industry is probably five to eight years away from the



NOTES:

(1) Theoretical limit, current wafer and feature sizes.

(2) Theoretical limit, assuming continued improvements in wafer size increase and feature size reduction.

(3) Theoretical limit, assuming constant rate of wafer size increase and an *increasing* rate of feature size reduction.

(4) Increase due to design and process innovations.

(5) Increase due to feature size improvement.

(6) Increase due to wafer size.

Data courtesy of T. Klein, National Semiconductor.

Figure 2—Past and projected contributions to batch density improvements

time when the cost and throughput of E.B. systems will allow practical use in direct wafer manufacture.

Feature size reduction, alone, does not determine the size of features that will be used, only the lower limit available. The devices from which the circuits are constructed must be scaled within other physical constraints.

Theoretically, MOS transistors, for example, can be scaled to quarter micron feature size, but some device characteristics don't scale well. As devices are scaled, circuit operating voltages and power densities must scale proportionately.

Such scaling requires re-engineering of device structures, process control, and film thicknesses. Parasitic parameters, such as leakage, noise, sub-threshold current, and sheet resistance do not scale as other circuit parameters and will require innovative design and/or materials research to deal with them.

Considering these factors then, it is theoretically possible for the batch density to increase by another 100 times in the next eight years. The task will require very large investments in capital and talent. The number of manufacturers able to make these investments will decrease. Moreover the market for memory products must continue to expand at a rate that will allow amortization of those investments.

It is appropriate at this point to apply this information to predictions of the coming generations of semiconductor memories.

In dynamic RAMs, MOS will dominate. The 16K will be the main production part through mid-1980. By mid-1980 significant quantities of 64K RAMs will be available. The 64K Dynamic RAMs will use scaled MOS devices (4 micron feature size), reduced power supply voltages (probably 5 Volts), $10f^2$ to $12f^2$ cell sizes, and be manufactured on 4 inch wafers. Some of these same techniques may be applied to the 16K RAMs but the cost of packaging four 16K RAMs will be offset by the lower cost of 64K RAMs, both for the manufacturer and for the user.

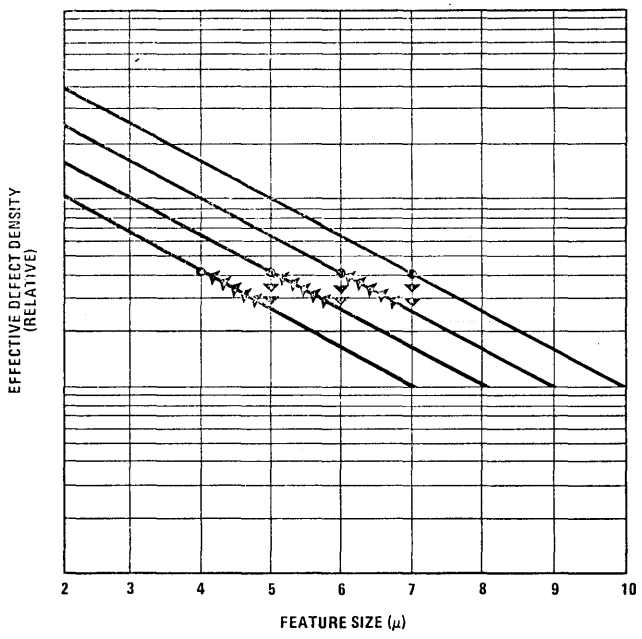


Figure 3—Effective density of catastrophic defects vs. feature size

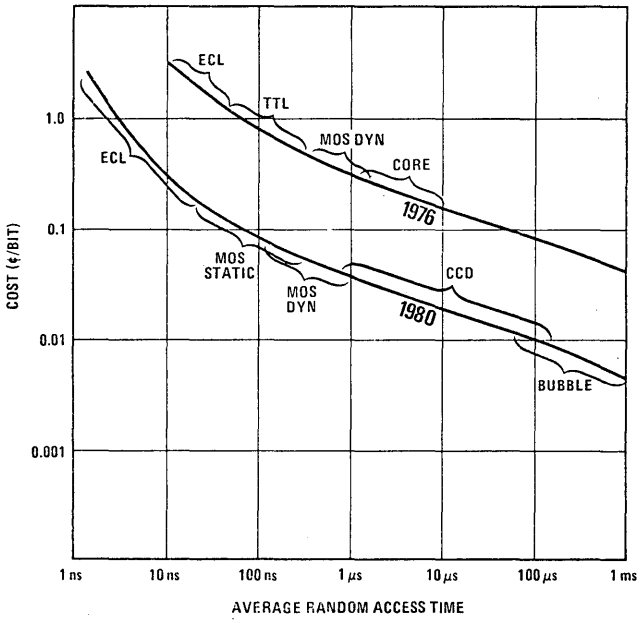


Figure 4—Memory component cost vs. performance

Storage cells for CCD memories are inherently easier to simplify and already 64K CCD's with $5⅓f^2$ cell sizes are close to reality. The disadvantages of serial memory storage will be offset by the lower cost per bit associated with CCD's. Memory system and CPU manufacturers will accept the 5 to 10 percent throughput penalties in exchange for ⅓ to ¼ the cost per bit CCD's will offer.

MOS static RAMs will use scaled MOS to achieve data rates presently served by bipolar technology. They will be used predominantly to serve the IBM Add-on business until the problems of interfacing Dynamic RAMs to the new generations of machines are solved. Currently, 4K static RAMs are becoming readily available. By mid-1980 16K static RAMs will be readily available. Present CPU's are implemented with ECL logic, and the translation from ECL to TTL levels to interface the RAMs adds appreciably to the access time. Future generations of MOS static RAMs will appear with ECL interfaces to eliminate the external level shifting and potentially reduce access times.

Bipolar static RAMs will continue to be the faster and smaller density memory devices. The bulk of the products will be ECL due to interfacing and access time requirements. Products will range from sub-five nanosecond 128-bit memories to sub-twenty-five nanosecond 4K bit memories. Cell sizes will be relatively large ($120f^2$) because of scaling difficulties and complex cell designs. Wafer size will be four inches. Device development will be concentrated towards reduction of parasitics (i.e., passive isolation) and increased gain band-width product of the transistors.

In summary, Bipolar static RAMs will move to higher speeds, MOS static RAMs will fill the speed range presently served by Bipolar RAMs (at lower cost and higher density), MOS Dynamic RAMs will be denser and lower in cost per bit, and CCD's will extend the range of semiconductor memories in the direction of longer access time and lower cost per bit. (Figure 4) The investments in terms of talent and capital will be large for the manufacturers able to stay in the competition.

Grateful acknowledgment goes to Thomas Klein for his analysis of Batch Density relationships.

Bubbles and CCD memories—Solid state mass storage

by J. EGIL JULIUSSEN
Texas Instruments Incorporated
Dallas, Texas

INTRODUCTION

In the last year significant technical advances have taken place in the development of magnetic bubble memories (MBM) and charge coupled devices (CCD). The first bubble memory chip has been introduced with bigger and better chips under development. Although 16K bit CCD chips have been available for two years, it is the recently announced 64K bit chips which will make an impact on computer system designs. The first commercial products based on MBM and CCD chips are now starting to appear.

With this progress it is worthwhile to look at the status of CCD and bubble memories, their applications and future potential.

MBM AND CCD STATUS

Both CCD and bubble memories are integrated circuit analogs to rotating electromechanical memories such as disks and tapes. CCDs store information in rows of capacitors. Bubbles and electromechanical memories store information as rows of magnets. The access methods are different. Electromechanical memories move the media while the magnetized regions remain stationary. In CCD and bubble memories the opposite takes place: the media remains stationary while the information, charge or magnets, move within the media. This avoids all mechanical motion and its associated drawbacks. The result is two solid state mass storage technologies—or electronic disks. For further information on the technical aspects of MBM and CCDs see References 1-10.

The features of MBM and CCDs are shown in Table I. Both technologies are serial access memories and are organized as shift registers. Bubble memories are nonvolatile as the information is retained without power. The electrical analogy of the magnetic bubble, the CCD, does not keep its charge or information as the power is turned off. CCDs are volatile and must be periodically refreshed to retain the information.

Both technologies have the modularity advantages arising from their chip packaging. The resulting low entry price is very important for small systems. As chips they can also be packaged directly with the CPU on PC boards.

The characteristics of today's major CCD and bubble memory components are shown in Table II. The 92K bit MBM chip has an average access time of 4 milliseconds and has a transfer rate of 50K bits per second. The dual-in-line package is larger than most ICs. The MBM package includes two permanent magnets giving non-volatility and two coil windings. The two coils produce a rotating magnetic field which moves the bubbles around the shift registers.

MBM support chips are already available. A minimum bubble memory system which is interfaced to a microprocessor can be made with 7 chips. The coil drivers produce the rotating magnetic fields. The sense amplifier detects the bubble information and is very similar to a core memory sense amplifier. The function timer and driver produce the necessary timing signals to control the operation of bubble memory chips. The controller chip is a LSI device which controls the memory system operations, buffers data and handles interfacing with a microprocessor.

The 64K bit CCD chips have a superior access time over bubble memories. The TI[9] and Fairchild devices have a 410 microsecond average access time and can transfer data at 1 to 5 Mbits per second rate. The Intel device has more shift registers with shorter length that gives better access time, but at a lower transfer rate. During operation the CCD chips use less power than bubble memory chips. The power advantage is reversed to MBMs in a standby mode.

None of the CCD products have any support chips. It should be noted that CCDs do not need an external sense amplifier. This function and the refresh function have been integrated onto the chip (one per shift register).

An assessment of MBM and CCD merits are shown in Table III. The main advantages of bubble memories are the nonvolatility and the large number of bits per chip. In effect MBM is the only solid state technology which is nonvolatile. The major advantages of CCDs are the low access time and the high transfer rate.

Bubble memories have an advantage over CCD from fewer manufacturing steps and fewer masking steps. This is also reflected in a potential higher MBM packaging density. However, the experience from MOS manufacturing is very applicable to CCD production and probably outweighs the

TABLE I.—MBM and CCD Features

MBM	CCD
• SERIAL ACCESS MEMORY	• SERIAL ACCESS MEMORY
• SHIFT REGISTER ORGANIZATION	• SHIFT REGISTER ORGANIZATION
• BLOCK ADDRESSABLE	• BLOCK ADDRESSABLE
• NONVOLATILE STORAGE	• VOLATILE STORAGE
• READ-MODIFY-WRITE	• READ-MODIFY-WRITE
• MODULAR STORAGE CAPACITIES	• MODULAR STORAGE CAPACITIES
– LOW ENTRY PRICE	– LOW ENTRY PRICE
– BIT PRICE INDEPENDENT OF STORAGE SIZE	– BIT PRICE INDEPENDENT OF STORAGE SIZE
• FEW MANUFACTURING STEPS	• TTL COMPATIBLE I/O
• MANUFACTURE SIMILAR TO IC PRODUCTION	• MOS COMPATIBLE MANUFACTURING
• STOP/START OPERATION	• STORAGE REFRESH REQUIRED
– LOWERS EFFECTIVE ACCESS TIME	
– LOWERS POWER DISSIPATION	
– VARIABLE TRANSFER RATE	
– MINIMAL DATA BUFFERING	

MBM advantage today. As bubble memory producers gain experience, it will be interesting to see if the potential MBM manufacturing advantages can be realized.

In packaging CCDs have an advantage due to the added complexity of the coils and magnets for a MBM chip.

In interfacing complexity MBM currently has an advantage from the availability of support chips. The low transfer rate and these support chips are very advantageous for microprocessor based systems. The high performance requirements of mainframe and supermini computers give CCDs a distinct interfacing advantage for these applications.

The bottom line is price. In the next few years, CCD will probably have an advantage due to MOS manufacturing experience and CCD availability from three or more manufacturers. With increased bubble memory manufacturing experience, MBM should close this gap and have an excellent chance of gaining an advantage over CCDs.

Table IV shows the announced MBM and CCD products the author is aware of. There are undoubtedly more products under development.

The three first MBM products use TI's 92K bit bubble memory chip. The TI 763/765 portable bubble memory terminals are part of TI's Silent 700®* series of terminals. Both terminals have a minimum of 20K bytes of bubble memory with optional expansion to 80K bytes.[11]

Q1 corporation has a microcomputer system with 80K bytes of bubble memory and also has a plasma display. Data Systems MBM system is a floppy disk replacement which is PDP-8 and PDP-11 plug compatible. Storage capacities of 86-519 Kbytes are available.

AT&T's 13A announcement system uses 68K bit bubble memory chips developed by Bell Laboratories and manufactured by Western Electric. The system stores pre-recorded messages which are repeatedly played back as telephone system messages. The system is being field tested by AT&T.[12]

The three CCD systems use Intel's 16K bit CCD chips. Intel's CCD board is an OEM product that can be used to develop end user systems. Technical Analysis Corporation has developed a fixed head disk replacement for their Nova based small business computer.[13] Alpha Data also has a

* ®Registered trademark.

TABLE II.—MBM and CCD Chip Characteristics

	TEXAS INSTRUMENTS		FAIRCHILD	INTEL
	TIB 0103 MBM CHIP	TMS 3064 CCD CHIP	F 464 CCD CHIP	2464 CCD CHIP
STORAGE CAPACITY	92 KBIT	64 KBIT	64 KBIT	64 KBIT
AVG ACCESS TIME	4 MSEC	410 SEC	410 SEC	130 SEC
MAX TRANSFER RATE	50 KB/SEC	5 MB/SEC	5 MB/SEC	2.5 MB/SEC
POWER	0.7 W	0.26 W	0.34 W	0.33 W
STANDBY POWER	NONE	25 MW	66 MW	41 MW
OPERATING TEMP	0 TO 70° C	0 TO 70° C	0 TO 55° C	-10 TO 85° C
PACKAGING	14-PIN DIP (1" x 1.1")	16-PIN DIP	16-PIN DIP	18-PIN DIP
PRODUCTION STATUS	PRODUCTION	PRODUCTION	PRODUCTION	PRODUCT ANNOUNCED
SUPPORT CHIPS AVAILABLE	CONTROLLER CHIP COIL DRIVER SENSE AMPLIFIER FUNCTION DRIVER FUNCTION TIMER	NONE	NONE	NONE

fixed head disk replacement which is compatible with their other fixed head disk products.

There are many products in the development stage which use the newer 64K bit CCDs, but none are announced yet.

To summarize the status of CCD and bubble memories a list of the companies with known R&D efforts are shown in Table V. As could be expected the major semiconductor manufacturers appear on the CCD list. Until recently only TI of the semiconductors producers had bubble memory effort. In the last year both Intel and National have started bubble memory efforts.

MEMORY SYSTEM TRENDS

It has been the goal for computer systems to have a single memory technology that has both the lowest cost and the highest performance. This goal has not been realized and the memory hierarchy from fast semiconductor RAMs to low cost disks and tapes must be used. The well-known memory hierarchy gap in access time and cost has provided a market pull for new memory technologies. In the early seventies this was the major driving force to develop CCD and bubble memories.

Additional market pull forces are now equally important. The proliferation of microprocessor applications are demanding small, non-volatile mass memory systems. Severe environments and the high maintenance costs need improved reliability. The popularity of virtual memory operating systems need some real and high performance mass memory.

The trend towards using fixed heads in combination with moving head disks is one solution that gives higher performance mass storage. However, there is still a large percentage of computer systems that are disk-bound.

MBM and CCDs are not removable storage technologies and this is a drawback vis-a-vis disks and tapes. There is a trend towards using non-removable disks. This lowers the disk drive cost and also improves reliability. The trend is also advantageous for both MBM and CCDs as both technologies can replace the function of non-removable disks. At the same time, the use of distributed processing systems

TABLE III.—MBM Versus CCD

	MBM	CCD
MAJOR ADVANTAGES	NONVOLATILE BITS/CHIP	ACCESS TIME TRANSFER RATE
MANUFACTURING	FEW STEPS FEW MASK LEVELS	EXPERIENCE FROM MOS MANUFACTURING
PACKING DENSITY	ADVANTAGE DUE TO MASK LEVELS	
PACKAGE	COILS & MAGNETS ADD COMPLEXITY	SIMPLE DIP
INTERFACING	ADVANTAGE FOR MICROPROCESSORS	ADVANTAGE FOR MAINFRAMES
PRICE	ADVANTAGE BASED ON PACKING DENSITY	ADVANTAGE FROM MOS MANUFACTURING EXPERIENCE

also lowers the need for removability. The data communication links in many cases are a substitute for removability.

To indicate how MBM and CCDs compare with other storage technologies, the system prices for end users are shown as a function of storage capacities in Figure 1. For small storage capacities MBM and CCDs have cost and physical size advantages. As soon as a few megabits are needed, the disks have a price advantage, but MBM and CCDs retain their performance edge. It is expected that MBM and CCDs, in the future, will be price competitive at higher and higher storage capacities.

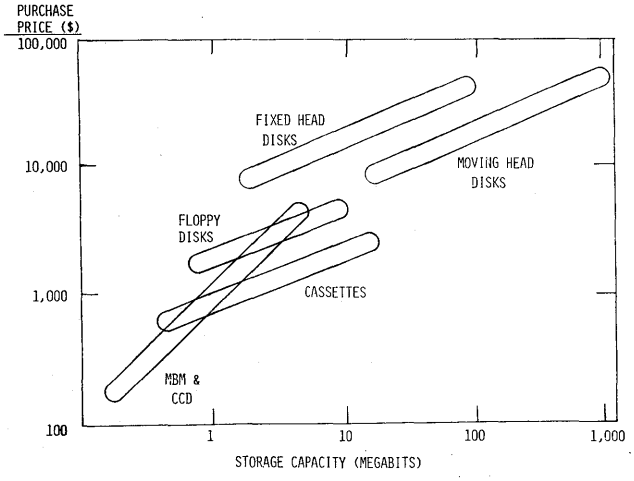


Figure 1—1978 memory system prices including interfacing

MBM AND CCD APPLICATIONS

To gain insight in the use of CCD and bubble memories Table VI shows the various storage peripheral product categories for mainframe, mini and microcomputers. MBM and CCDs are not fast enough to be used as main memory, and are too costly to be used for removable mass storage. As fast auxiliary memory (FAM) both MBM and CCDs have excellent credentials. This is the product category which fills the memory hierarchy gap. In the non-removable mass storage category, MBM and CCDs are very viable, but will see strong competition from disks.

Due to their packaging flexibility, MBM and CCDs add an extra dimension to mass storage applications. They can be packaged directly with the processor and become an integral mass storage device. No extra box is needed as is

TABLE IV.—Announced MBM and CCD Products

MBM

- TEXAS INSTRUMENTS
  - TI 763/765 PORTABLE BUBBLE MEMORY TERMINAL
  - 20 TO 80 KBYTES MBM
- QI CORPORATION
  - QI/LITE MICROCOMPUTER
  - 80 KBYTES MBM
- DATA SYSTEMS
  - DSD 640 FLOPPY DISK REPLACEMENT
  - 86 TO 519 KBYTES MBM
  - PDP-11 & PDP-8 PLUG COMPATIBLE
- A T&T
  - 13A ANNOUNCEMENT SYSTEM FOR RECORDED SPEECH
  - 68 KBYTES OF MBM FOR 24 SECONDS OF SPEECH DATA

CCD

- INTEL
  - IN-65 CCD BOARDS
  - 128K TO 1 MBYTES CCD
- TECHNICAL ANALYSIS CORP.
  - FIXED HEAD DISK REPLACEMENT
  - 256K TO 2 MBYTES CCD
  - DG NOVA PLUG COMPATIBLE
- ALPHA DATA
  - CCDISC - FIXED HEAD DISK REPLACEMENT
  - 128K TO 1 MBYTES CCD

usually the case with disks and tapes. Instead the MBM or CCD can be put on the same or different PC board as the processor. The TI 763/765 portable bubble memory terminal is an excellent example of this advantage.

The most likely CCD and bubble memory applications are shown in Table VII. The main bubble memory applications are in small systems. The bubble memory characteristics fit microperipherals and integral mass storage applications very well. In the miniperipheral category, MBM add-in or add-on electronic disks are viable opportunities. Add-in refers to MBM packaged with the CPU, while add-on means a separate box. In essence MBM is a main candidate for distributed mass storage to go along with distributed processing.

The main CCD applications are in closing the memory hierarchy gap. These opportunities are mostly in the large computer market place. In other words, CCD will be the technology which makes virtual memory systems become real.

With MBM and CCDs having similar features, will they compete head-on for the same applications? To answer this question, Table VIII shows the probable MBM and CCD usage.

The high performance characteristics will favor CCDs in mainframe computer applications. CCDs for fixed head disk replacement have already seen usage. Virtual memory and moving head disk cache memory based on CCDs are probably in development.

The nonvolatility and support circuit availability of bubble memories will favor MBMs as integral mass storage for microprocessor based systems. The announced MBM products fall in this category, and more are in the development stage.

The effect is that MBM and CCDs will have little direct application competition.

In summary, the usage of MBM and CCDs will evolve as follows: In 1977 many system designers gained familiarity with the two technologies. In 1978 CCD and bubble memories are designed into many products. In 1979 numerous products with MBM and CCDs will be in production. In the 1980's both MBM and CCDs will have become important storage technologies.

TABLE V.—Companies with MBM or CCD Effort

MBM

- US COMPANIES
  - TEXAS INSTRUMENTS
  - BELL LABORATORIES
  - IBM
  - ROCKWELL INTERNATIONAL
  - HEWLETT-PACKARD
  - UNIVAC
  - INTEL
  - NATIONAL
- FOREIGN COMPANIES
  - FUJITSU
  - HITACHI
  - NIPPON ELECTRIC
  - PHILIPS
  - PLESSEY

CCD

- US COMPANIES
  - TEXAS INSTRUMENTS
  - BELL LABORATORIES
  - IBM
  - FAIRCHILD
  - INTEL
  - NATIONAL
  - MOSTEK
  - MOTOROLA
  - HUGHES
  - RCA
- FOREIGN COMPANIES
  - BELL NORTHERN
  - PHILIPS
  - PLESSEY
  - SIEMENS
  - TOSHIBA
  - NIPPON ELECT
  - MITSUBISHI

TABLE VI.—Storage Peripherals

	MAIN MEMORY	FAST AUXILIARY MEMORY	NONREMOVABLE MASS STORAGE	REMOVABLE MASS STORAGE	
				ON-LINE	OFF-LINE
PERIPHERALS	CORE MOS RAM BIPOLAR RAM	FIXED HEAD DISK EBAM CCD MBM	MULTIPLATTER DISK EBAM CCD MBM	MULTIPLATTER DISK	AUTOMATED TAPE SYSTEM MAGNETIC TAPE
MINI-PERIPHERALS	CORE MOS RAM	FIXED HEAD DISK CCD MBM	SINGLE PLATTER DISK MBM CCD	SINGLE PLATTER DISK FLOPPY DISK	MAGNETIC TAPE TAPE CARTRIDGE FLOPPY DISK
MICRO-PERIPHERALS	MOS RAM ROM PROM EPROM	MBM CCD	MBM CCD	MINIFLOPPY	CASSETTE MINIFLOPPY MINICASSETTE

TABLE VII.—MBM and CCD Applications

MBM	CCD
• MICROPERIPHERALS	• MAINFRAME PERIPHERALS
◦ MICROCOMPUTER MASS STORAGE	◦ VIRTUAL MEMORY
◦ FLOPPY DISK REPLACEMENT	◦ DISK CACHE
• INTEGRAL MASS STORAGE	◦ FIXED HEAD DISK REPLACEMENT
◦ PROGRAMMABLE TERMINALS	• MINIPERIPHERALS
◦ PORTABLE TERMINALS	◦ ELECTRONIC DISK
◦ WORD PROCESSING TERMINALS	◦ OPERATING SYSTEM STORAGE
◦ BANKING TERMINALS	◦ VIRTUAL MEMORY
◦ POINT-OF-SALE TERMINALS	◦ FIXED HEAD DISK REPLACEMENT
◦ DATA COLLECTION TERMINALS	• MICROPERIPHERALS
◦ PROGRAMMABLE CALCULATORS	◦ MICROCOMPUTER MASS STORAGE
◦ MEASUREMENT & TEST EQUIPMENT	◦ FLOPPY DISK REPLACEMENT
• MINIPERIPHERALS	• BUFFER STORAGE
◦ ELECTRONIC DISK	◦ CRT REFRESH
◦ FIXED HEAD DISK REPLACEMENT	◦ HIGH SPEED BUFFERING
◦ OPERATING SYSTEM STORAGE	
• MAINFRAME PERIPHERALS	
◦ FAST AUXILIARY MEMORY	
◦ DISK CACHE	

WHAT'S AHEAD IN CCD AND MBMs

CCD and bubble memories will follow the rapid increase in storage capacities and with the resulting decrease in cost per bit which is the characteristic of semiconductor technologies. Every two to three years the storage capacity of semiconductor chips quadruple and similar price per bit reductions take place. Both MBM and CCD chips will improve in this manner. In the beginning new storage technologies improve even faster and CCD and MBMs are likely to do so.

An overview of solid state memory trends is shown in Figure 2.[14] It shows that a 256K bit bubble chip can be expected in a year with a 256K bit CCD chip following a year later. By 1980 a 1 Mbit bubble memory chip is forecasted with the 1 Mbit CCD chip following a year later.

In terms of performance, CCD chips will still have an edge in access time and transfer rate over bubble memories. Both technologies will have higher transfer rates than today. The access time will be a trade-off parameter. To make larger chips will require longer shift registers. This would result in longer access time, unless a faster shift rate is used. It appears that for both MBM and CCD's the increased shift rate will just keep up with the longer shift registers. The result is that the access time of MBM and CCDs will be about the same as they are now.

But how will MBM and CCDs stack up against disks? The moving head disk technology has had impressive advances in the last 15 years, and will probably keep on improving

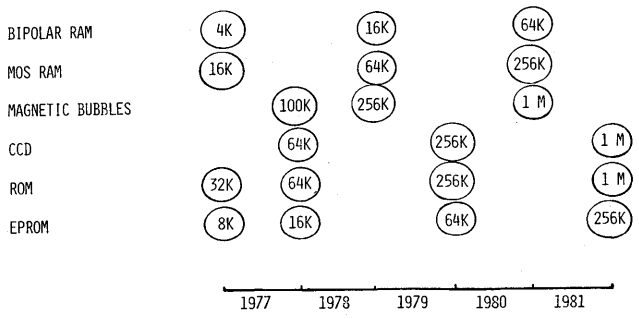


Figure 2—Solid-state memory technology trends

TABLE VIII.—MBM and CCD Usage

	MBM	CCD	COMMENT
MAINFRAME COMPUTERS	FHD REPLACEMENT IS A POTENTIAL APPLICATION ELECTRONIC DISKS AWAIT MBM PRICE REDUCTION	FHD REPLACEMENT IS AN IMMEDIATE APPLICATION VIRTUAL MEMORY AND MHD CACHE WILL EMERGE	CCD'S MAJOR NEAR-TERM APPLICATION
MINICOMPUTERS	SMALL ADD-IN ELECTRONIC DISK WILL EMERGE ELECTRONIC DISKS COMPETITIVE WITH CARTRIDGE DISKS MAY APPEAR BEFORE 1980	SMALL ADD-IN FAMS WILL SOON BE AVAILABLE	APPLICATION FOR BOTH MBM AND CCD
MICROSYSTEMS	INTEGRAL MASS STORAGE FOR MPU-BASED SYSTEMS IS AN IMMEDIATE APPLICATION	INTEGRAL MASS STORAGE FOR SOME APPLICATIONS	MBM'S MAJOR NEAR-TERM APPLICATION

for another five to ten years. The disk advances have been realized by packing an increasing number of bits in the same area. The result is that large multi-platter disks that stored 2M bytes in the early 1960's now store 300M bytes, and cost about the same. Simultaneously lower priced units have been developed. The cartridge or single platter disk appeared in the late 1960's and now stores about 100M bits. The floppy disk was introduced in the early 1970's and just 2 years ago the minifloppy disk was introduced.

Impressive as these improvements are, the CCD and bubble memories will become increasingly competitive with disks. The disk technology is vulnerable because its bit price is very dependent on the storage capacity. The bit price of a minifloppy is about 50 times higher than the bit price of large disks. By 1980-81 a single MBM or CCD chip will store more than today's minifloppy disk. The single unit OEM price of the minifloppy is about $350 today and may decrease another $100 in the next couple of years. This will be no match for the learning curve pricing of MBM and CCDs. An analogy of what is likely to happen is the microprocessor price which has decreased from several hundred dollars to less than ten in just three years.

In summary, it appears that CCD and bubble memory systems will be price competitive with floppy and cartridge disks by the early 1980's.

SUMMARY

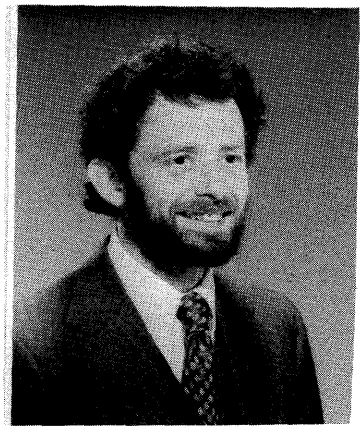
For a new storage technology to succeed it must meet several requirements. The technology must have advantageous features and characteristics including competitive price/performance. Long term technology improvement potentials must exist. Significant R&D investment in the technology by many companies is crucial. There must, of course, be available markets. Equally important is an initial market entry niche.

Both CCD and bubble memories meet these requirements and should therefore succeed in becoming major storage technologies in the 1980's.

REFERENCES

1. Bobeck, A. H., et al., "Magnetic Bubbles—An Emerging New Memory Technology," *Proceedings of the IEEE*, August 1975.

2. Juliussen, J. E., "Magnetic Bubble Systems Approach Practical Use," *Computer Design,* October 1976.
3. Lee, D. M., "Bubble Memory for Microprocessor Mass Storage," IEEE Compcon, Spring 1977, February 1977.
4. Juliussen, J. E., "Bubble Memory as Small Mass Storage," Electro 77, New York, April 1977.
5. Bobeck, A. H., "The Development of Bubble Memory Devices," Electro 77, New York, April 1977.
6. Myers, Ware, "Current Developments in Magnetic Bubble Technology," *IEEE Computer,* August 1977.
7. Carnes, J. E., et al., "Charge-coupled Devices for Computer Memories," *Proc. National Computer Conference,* 1974.
8. Bhandarkar, D. P., "Digital CCD Memory Trade-offs: A System Viewpoint," *Microelectronics,* Vol. 7, No. 2.
9. Barton, J. B., et al., "A Cost Effective 64K CCD Memory," Electro 77, New York, April 1977.
10. Bhandarkar, D. P., "Dynamic MOS Memories: Serial or Random Access," IEEE Compcon Spring 1978, February 1978.
11. Flannigan, J. S., "Bubble Memory Terminal, An Added Dimension to Data Entry," IEEE Compcon Fall 1977, September 1977.
12. Williams, J. E., "Magnetic Bubble Memory in Telephone Systems," Electro 77, New York, April 1977.
13. Jamieson, J. M., "The CCD Memory Gains a Foothold," Mini-Micro Systems, April 1977.
14. Juliussen, J. E. and W. J. Watson, "Problems of the 80's: Computer System Organization," The Oregon Report on Computing, Portland, March 1978.



Area Director:
Michael A. Harrison
University of California
Berkeley, California

Programming and operating systems

Four very different aspects of the areas of programming languages and operating systems are represented in these sessions.

A unique session will be a report on the recent History of Programming Languages Conference. No single topic seems to generate as much varied and heated debate as the merits of a programming language. Since future generations learn of past events through historical documents, special care had to be taken in organizing a History of Programming Languages Conference to insure a balanced and fair treatment of the various topics. An attempt will be made to summarize the technical lessons learned.

The 1970's have seen increasing interest in secure computing systems. Both theoretical results and practical experience with our systems seem to be somewhat discouraging. Nonetheless, our intrepid designers keep trying. A report will be made which focuses on how close we are to achieving such systems.

There has been a flurry of recent work on cryptography which suggests that improved and economical ciphers are available. Different aspects of such work will be discussed. One is the NBS standard while another concerns the use of data dependent keys.

There will be a session of the status of COBOL which will discuss, among other things, the anticipated 1980 standard and also data base interfaces.

Issues in kernel design*

by GERALD J. POPEK and CHARLES S. KLINE
University of California at Los Angeles
Los Angeles, California

INTRODUCTION

As operating systems became larger and more complex, interest increased in segmenting that software in a rational fashion. With respect to operating systems, a number of efforts were made to develop a low level base that would provide basic system functions in a highly reliable way. On top of this skeletal software base, extensive operating system functions and user supports would be built. Early efforts in this direction, although significantly different in scale and goals, include IBM's virtual machine system CP-67,[1] Brinch-Hansen's RC-4000 nucleus,[2] and CAL-TSS.[3]

However, the greatest impetus to this activity grew recently out of the desire for secure operating systems—those that could assure that the access to data stored within was controlled and protected in an uncircumventible way. Efforts have been made by a number of groups to design and develop operating system bases that, in addition to providing necessary primitive functions, were wholly responsible for the security of the operating system, including whatever was built on top of that nucleus. Part of the goal of these efforts typically was to minimize the size and complexity of the resulting nucleus, in the well founded belief that to do so would greatly increase the likelihood that the resulting software would be correctly implemented. Desires to apply program verification methods to this software have heightened the importance of these minimization goals, since current verification methods are applicable only to modest sized, simply structured programs. Such a minimum size, security oriented operating system nucleus is typically called a "security kernel."[4]

It should be noted that these efforts have taken place in an environment where reliable security has become of increasing concern. No general purpose operating system of a more traditional design has successfully provided a satisfactory degree of assurance for those who are seriously concerned about control over their data. Known flaws are so numerous that it is generally recognized a highly systematic approach is required. Since it appears that kernel based architectures will make considerable strides in improving that situation, an understanding of their characteristics is useful.

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-C-0211.

There are several considerations that make a kernel based operating system architecture significantly different from a more traditional architecture developed without security as a major design criterion. Reliable security demands that the software upon which security depends be as small and simple as practical. As a result, functions included in operating system bases to enhance performance or increase the convenience of writing software above that base are not needed for security purposes and consequently not included in a kernel. Kernel designs would typically exclude that software despite potentially increased performance overhead or greater non-kernel software complexity. Naturally, the architecture is structured to avoid these costs as much as possible, and it appears that such penalties can generally be minimized.

Conversely, the kernel must contain all security related software if correct enforcement is not to depend on non-kernel software. Therefore, as a countering influence, functions are forced into the kernel that may have been omitted from an operating system base and implemented at an outer level. The body of this paper characterizes the architectural principles that lead to these changes. Examples of the relocation of system functions in an architecture are also given.

EFFECTS OF DESIGN CONSTRAINTS ON KERNEL ARCHITECTURES

While kernel architectures in general exhibit many common characteristics and differ substantially from traditional operating system architectures, they also differ significantly among themselves. The technical constraints concerning security policy, function, hardware and performance must be specified before it will be possible to develop appropriate kernel specifications. Each of these considerations is discussed in turn below.

Security policy

The amount of mechanism that must be included in a security kernel is greatly affected by the particular security policy (control discipline[5] or level of function[6]) that it is desired to enforce. For example, the support required for

isolation is considerably less than what is needed for general support of intimate controlled sharing with domain switches within an individual process.

Kernel architectures have been directed primarily at the goal of reliable "data security": assuring that it is possible for users only to access data to which they are specifically entitled. Issues such as confinement or denial of service have generally not been directly addressed, although care usually has been taken to minimize such problems. Clearly, the more sophisticated the security policy that it is desired to enforce, the more kernel mechanism will likely be needed.

Data security policies themselves may be subdivided on a number of criteria. What are the objects that are protected? An operating system kernel might support any one of the subsets of an extensive list of potential objects: processes, pages, segments, files, file subtrees, messages and/or message channels, records, fields, mini-disks, entire devices (tape drives, disk packs, terminals) or partitions of devices (blocks on a tape, windows on a crt terminal).

Next, what is the *object grain:* the size of the objects protected. Are processes the active objects, or can a procedure call within a process coincide with a domain change, as in Hydra.[7] Perhaps, in a system that supports a family tree of processes for a given user, the entire family is the active object, with no distinction made among processes. That is, all processes in the tree might be required to have the same access rights. This design was used in the TENEX operating system.

What are the rules governing the alteration of system data that records the security policy? That is, what is the *policy grain.* Is control over objects and groups of objects hierarchically distributed as in Multics,[8] or is it strictly centralized, as in the IBM operating systems. What is the precision with which one can specify access control decisions? Can a file be marked with the exact list of users to be given access, or can one only set the access mode as either public or private, to illustrate two extremes.

System functions

Apart from the question of the security policy supported by a kernel, a serious impact on the actual kernel design and implementation results from the functions and services to be supported by that kernel. Is it possible, for instance, to create and destroy objects of all types supported by the kernel, or do certain cases, such as mini-disks, have their allocation fixed at the time the kernel is assembled. To what degree is the number of objects of each type supported and protected by the kernel fully variable. How rich are the set of operators provided to access the kernel protected objects. Are sophisticated message synchronization primitives built, or is a simpler, less efficient facility provided, out of which user software must simulate the desired features.

Type extensibility is another aspect of system function. Is it possible for user programs to create new objects of new types and have these new objects protected in a fashion similar to previously existing objects. This type extensibility first appeared in programming languages, and subsequently in the CAL-TSS system.[3]

Another aspect of system functionality that invariably affects a security kernel concerns device support. For reasons discussed later, most hardware designs require, for security to be reliably enforced, that there be additional kernel software for each device type. Therefore, the more general hardware base to support, the larger and potentially more complex a kernel can result.

Another significant consideration affecting a security kernel is the convenience with which operating system software can be constructed on top of the primitives which are provided. If certain facilities are entirely absent, selected functions may be impossible to provide. The existing Unix kernel does not permit asynchronous I/O within a user process.[9] Other functions may be possible to build, but only in a relatively inconvenient way. The UCLA kernel[17] does not support the process hierarchies required by UNIX, although they can be built outside the kernel using inter-process communication facilities.

In general, one wishes as little functionality in the kernel as is sufficient. In applying this philosophy, one should examine the necessary functions and determine whether a different, smaller and simpler set would suffice. For example, extensive real time functions, synchronization facilities, and interprocess communication mechanisms may be desired. However, the base system might be carefully designed only to include a small set of well chosen I/O primitives, augmented by a signalling call and a mechanism by which a user process could mark itself uninterruptible for a short period. The extensive set of user desired functions might then be constructed in application code using the simple base facilities.

Hardware effects

The hardware base on which a kernel is built can significantly impact the resulting kernel, including its size, complexity, overall architecture, and details of implementation. The richness of the hardware base that must be supported will have considerable effect. A multiple processor system, in which more than one cpu executes kernel functions, may require support for processor coordination within the kernel that single processor systems do not. Certain types of I/O channels may need detailed cpu software support.

Typically, the accesses made by central processors are constrained by memory management units, protection keys, or other hardware support. That support has the characteristic that privileged cpu software can set the controls and then permit untrusted software to run with assurance that it is not possible for software to exceed those access rights recorded in the hardware controls. The hardware implemented control features associated with channel accesses to main memory and to secondary storage are typically much less convenient, or even non-existent. Channels on most machines use absolute addresses to access main memory, instead of having a virtual reference modified and controlled

by relocation hardware. Therefore, unless some other form of hardware enforced protection, such as IBM's storage keys, limit the channel's access, additional kernel software is required to check and perhaps modify the addresses used by the channel in performing I/O. This task can be quite complex and error prone, even on IBM hardware, as reported by Belady and Weissman.[10]

While in some existing machines, channels' main memory access is controlled by programmable hardware assists, it is even rarer also to find hardware support for control of access to the secondary storage connected to the channel. Therefore it is generally necessary for kernel software to check or modify the secondary storage addresses and relevant commands that compose channel programs. Certain machines such as the IBM 360/370 series have some limited hardware supported control over secondary storage access. There, changes in cylinder or track can be blocked for the duration of a channel program's execution. However, such facilities are generally insufficient for direct use by the kernel to protect pages, segments or file systems.

I/O is clearly one area where hardware characteristics considerably affect the security kernel, but it is not the only one. A simple example, the time of day clock, on many machines is located in a way so that only privileged software can read it, despite the fact that the time of day is not relevant to most definitions of data security. On the PDP-11, both the word size and cpu arithmetic are organized on a 16 bit basis. Absolute addresses are larger however. I/O device registers (each 16 bits) have the additional bits located in additional registers, sometimes with little commonality among devices where those bits are to be found. Sometimes carry from the 16-th to 17-th bit is implemented, sometimes not. As a result, special software for each device is typically needed to check address bounds. Additional kernel code is required that could have been eliminated with judicious hardware planning.

There are often a number of such hardware characteristics that considerably complicate the task of building a secure kernel, and that would not require significant changes to greatly diminish those difficulties. The previous examples were largely of this class. More dramatic changes to the hardware base could of course greatly contribute to reliable system security. Capability based architectures have considerable promise in that respect. Kernel designs such as those being pursued at UCLA particularly lend themselves to implementation in firmware. The attractive feature about many of these points is that their adoption should not necessarily imply significantly increased fabrication costs, and both the simplicity and reliability of software would be enhanced.

Performance

The last significant issues which affect the size and complexity of the kernel are performance constraints. Stringent performance requirements as expected make it difficult to develop a straightforward design and implementation. For example, while it may be possible for scheduling decisions to be made through a separate user process which is only interrogated by the kernel, the additional process switch overhead to invoke the scheduler for each expected user process switch may impose unacceptable delays. Expensive domain changing usually exerts pressures that lead to larger, more complex kernels, since one is tempted to integrate such functions as scheduling into kernel code.

The costly effects of this phenomenon are well illustrated by the Multics experience.[11] On its original GE 645 hardware base, ring crossings (domain changes) were quite expensive, since the existence of rings was simulated in software by a complete switch of segment tables and status at each ring crossing. As a result, performance pressures led the highly privileged ring zero to be composed of approximately 80,000 lines of PL/1 code, and a number of security errors. Subsequent research has shown that a majority of that code can be removed.[4]

Even if performance requirements do not alter the functional specifications of a kernel, complexity may be increased by the need to code certain functions in a highly efficient manner. It may be necessary to replace a simple sequential search by a more complex hashing algorithm that uses chaining in a overflow area for collision resolution, as just one example.

These issues of security policy, system function, hardware impact, and performance requirements all affect the size and complexity of a security kernel which can be built. Within any given set of such constraints, however, certain designs are far superior to others. The next section of this paper discusses kernel design principles that generally seem to apply to most of the likely sets of constraints.

PRINCIPLES OF KERNEL DESIGN

A number of design principles have been developed as guides to the design of secure systems, notably "least privilege" and "least common mechanism". See Saltzer[6] or Popek.[8] However, they are little help without explanation of their architectural effect, examples of their application, or discussions of their effect on such operating system problems as resource management or performance.

Here we first make several general observations regarding kernel designs and illustrate specific tradeoffs with examples from existing systems. Most of these observations and illustrations are concerned with the tasks of developing as small and simple a kernel as is practical, under the design constraints discussed earlier.

Overall system architecture

One of the first questions concerns the relationship of the kernel to the rest of the operating system. There are a number of possibilities. In both the Multics system and a Unix prototype under development at the Mitre Corporation, the kernel is essentially part of the user process, even

sharing the process address space in the Multics case. Kernel code and data are protected from user process software by a variety of mechanisms. Kernel tables are shared among all processes. Interrupts may occur at nearly any time, even while running kernel code. Typically, at that point the current process can be suspended and a process switch performed. Therefore kernel data and tables must be designed for parallel access and modification.

An alternate approach, examined at UCLA, more completely separates the kernel from all other software. As in the RC 4000 kernel, the UCLA security kernel is not part of any process. It is run in its own address space, and is better thought of as a hardware extension. That is, each software implemented primitive instruction provides some security relevant function, and runs from invocation to completion without interruption. Functionally it acts as one hardware instruction. Therefore, a process state that indicates running in a powerful mode is never saved or restored, and parallelism complexities are considerably reduced or eliminated. While such an approach may reduce kernel complexity, it requires care in the overall architecture if adequate functionality is to be obtained. The values of each of these approaches are discussed in more detail elsewhere.[12]

Resource pools

One of the most effective ways to reduce the size and complexity of an operating system kernel is to remove, as much as possible, all of the resource management functions for a particular type of resource and relegate those functions to distrusted code. The degree to which this goal can be attained, of course, will vary from system to system, and from one object type to another within a given system.

There are three significant aspects to resource handling in operating systems that are subject to removal from an operating system kernel and supportable by user software. They are type integrity, management policy, and naming. Each is discussed in turn below.

Type integrity

The most extreme action possible is to remove the entire resource as well as all of its supporting software from the base level of the system. As a result, the kernel does not provide protection for these resources as such. Any structure which is needed to insure the type integrity of the resource is the responsibility of the user software. The only protection enforced by the kernel is that provided for the kernel recognized object in which the user implemented resources are placed: segments, pages, disk blocks and the like.

What this strategy generally means is that a single common pool of resources, usually maintained for all processes by the operating system, has now been broken up into a number of smaller subpools, each managed independently. This approach is illustrated by the I/O buffer management in OS/360. Buffer integrity is not assured, since whatever pointer the user supplies in certain SVCs is used by the operating system so long as the indicated transfer involves that user's core.

The tradeoffs involved in pursuing this design are straightforward. By moving all of the resource handling out of a kernel, including the maintenance of the structure or type of the resource, the kernel is further simplified, sometimes considerably. The performance cost is clear from elementary queuing theory. Separate subpools are in general more poorly utilized than the same number of resources in a single common pool, because some subpools may contain idle resources while others are exhausted and unable to fulfill requests directed at them. Alternately, additional resources will be required to maintain the same level of service. A designer must decide what the cost of additional memory or other resources is, or how much delay in response can be tolerated in return for what is likely to be a more secure system.

There are many resources that an operating system typically manages that are not so obvious as I/O buffers. Name spaces are a good example. There are a number of cases where unique names are needed for convenience in communication, like socket numbers in the ARPA net, which are usually assigned and reclaimed by the operating system. However, they could be preallocated to various domains, with each domain expected to do its own management. A kernel, using lower level names, can assure that messages are properly delivered to the appropriate domains in the ARPA net example, but need not be concerned with management of sockets.[13]

Resource management

In those cases where it is not possible to completely exclude support of a resource type from the kernel, it may be possible to remove the management of the resource, but leave that portion of the software responsible for the integrity of the resource objects. That is, the mechanisms which actually operate on the resource are part of the kernel, but the policy software which decides which operations should be performed, and under what circumstances, is distrusted, and part of user processes. This approach, which leaves type integrity as a kernel responsibility but resource management relegated to user code, is essentially the distinction made by Wulf between mechanism and policy.[7]

A good example of the application of this approach occurs in the UCLA kernel architecture.[14] There, process scheduling as well as the management of main memory is the sole responsibility of one particular user level process, containing distrusted code. It issues kernel calls to switch processes, initiate page swaps, and so forth. While this scheduling process therefore can move around the objects that it has the responsibility to manage, it cannot examine any of their contents. This method supports usual functionality requirements and permits sophisticated scheduling, while it considerably simplifies the kernel. However, certain confinement problems are not solved, as discussed later.

Naming

As pointed out by Janson,[15] a considerable amount of code in an operating system is devoted to managing the names of resources, in addition to the management of the resources themselves. A resource may in fact be known by several names, at differing levels of abstraction. In addition to enforcing conventions such as uniqueness of names, the mapping of one name to another must also be done. An example of this mapping can be seen in the handling of segments in Multics. Character string names for segments are used by the file system software (and by users) to specify a segment (analogous to a file). Segment numbers, indexes into a "Known Segment Table," are used by running programs to refer to active segments. The Known Segment Table of that user contains the physical address of the page table for the corresponding segment. Thus there are three different name spaces used to refer to segment resources, and mappings among these spaces must be maintained correctly.

The more of this name management that can be removed from the kernel, the simpler the resulting kernel software. There are several ways to do so. One need not fully support in the kernel each of the name mapping levels. Several of them, including their mapping software, could be built as part of user domains. This approach generally amounts to partitioning the higher level name spaces, with separate partitions managed by separate processes. However, such a design requires care if controlled sharing is supported by the architecture. User software certainly needs to know the names maintained by other user software in order to coordinate references to shared objects. The coordination can be accomplished if each name maintenance package follows the same conventions, and messages are used to exchange relevant information.

Of course, it is true that in any system some conventions must be externally agreed upon, and some information externally exchanged between people, in order to get coordination started, even if it is only where (in what directory perhaps) to look to find relevant information. Here it is being suggested that much of this coordination and convention handling need not be part of the security enforcement of an operating system. The one potential limitation in this viewpoint, concerning so called Trojan horses, is discussed in a later section.

One should note that it is possible to move only a portion of the name support out of the kernel, too. One might partition the name space, allocating each partition to a different user process, and then have the kernel merely enforce that partition. Each user would do his own name management within a given partition. This view is similar to that of type integrity, discussed earlier, with the resource being the names, and the operations implemented in the kernel which operate on names being very limited.

DECOMPOSITION OF SECURE CODE

Up until this point we have maintained the fiction that the appropriate system architecture for highly reliable security placed all security relevant code in the core of the operating system, the kernel, running on the bare hardware. The rest of the system software is encapsulated within user processes, running under the control of the kernel, and making calls to it.

Trusted processes

However, this structure is not the best, as recognized by work at M.I.T.,[11] UCLA,[12] SRI,[16] as well as in specifications for the military network SATIN-IV. Rather, there are a number of security relevant functions that can profitably be moved out of the kernel and placed in so called "trusted processes". These processes architecturally are the same as user processes, except that the information they pass to the kernel is used by the kernel to take security relevant actions.

A good example of such a trusted process appears both in the Multics system at M.I.T. as well as in the UCLA Secure Unix development. One process has responsibility for handling free terminals, logging and authenticating users, and starting up the appropriate software for them. The Multics logger and the UCLA initiator both run as standard processes, except that the Multics ring zero software and the UCLA kernel both accept the authentication decision made by that process, and make security relevant decisions using that information.

Levels of kernels

One sometimes finds that the software composing these trusted processes can be segregated into security relevant and nonsecurity relevant parts. That is, the trusted processes can often themselves be structured so as to be composed of a kernel and other, distrusted software.

The degree to which this segregation is successful can be increased if a few redundant checks are made. For example one might build a sophisticated hashing algorithm with clever collision resolution for password checks, but after the appropriate entry in the hashing table has supposedly been found by distrusted code, the table index can be given to a trusted subroutine to make the actual check and pass information to the operating system kernel. In VM/370, channel program translation is followed by a security check of the resulting code.

The net result of this segregation is a system composed of levels of kernels. Each kernel depends, for its correct operation, only on the correctness of its own implementation, together with that of the lower level kernels that provide security relevant functions. In this way, the necessity that there be security relevant, trusted code at various levels in a large system is met, while at the same time the total amount of code upon which the overall system's security depends has been minimized. Note that this viewpoint differs markedly from that taken at SRI, in which it is argued that the entire operating system design and implementation should be certified or verified.[16]

A more powerful example of the utility of a second level kernel concerns name mapping and file systems. There, conditions occur under which the user needs to refer to

system supported objects by names that are from a different name space than that supported by the kernel. For example, as already mentioned, the kernel may maintain segment numbers, while the user will invariably wish to refer to those segments by character string names, especially when setting access control information, but perhaps also when stating which segment to read or modify. Because of these usage patterns, mapping between name spaces may be highly security relevant. File systems provide an important case of this problem. In the UCLA Unix system, file management is done by a user level process.[17] The kernel of that file manager is responsible for elementary directory management, and other distrusted software in the process can take care of positioning pointers within files, scheduling I/O operations and maintaining resource allocation limits. The directory management consists largely of mapping string names to disk block names supported by the kernel, handling allocation matters, and checking and maintaining protection data.

This multiple level kernel architecture can yield significant advantages in the design and implementation of the base level kernel. If applied in conjunction with other design principles discussed above, the base level kernel can become little more than an implementor of abstract types. It makes segments, processes, messages, and so forth out of hardware memory, status registers, and the like. For some systems, this simplification of the base level kernel may permit further improvement. It may be possible to implement each kernel call (including the interrupt handler, which is just a hardware forced call) as uninterruptible code, without losing any relevant performance characteristics. This design virtually eliminates any concern for parallelism in the security relevant code, and therefore enhances its reliability. If the kernel is simple enough, it is a candidate for implementation in microcode. A packet switch communications processor that hosts no user programming is an example of an application that might only require a kernel simple enough for such an implementation.

A final note on levels of kernels concerns the updating of the protection data used by trusted software to make access control decisions. If a user at a terminal is to have confidence in the security enforced by a system, he must be able to view and change the system's protection data in a highly reliable way. He needs a secure channel between his terminal and kernel code, without any distrusted software intervening, even to manage character handling. One easy method to implement that channel is to provide a simple facility by which the user can switch his terminal to a trusted process and back. In this way he need not be concerned with any software running in his own process(es). While this solution, unless extended, suffers from the inability to permit program generated changes in protection data, such actions do not seem to occur often in practice anyway, except as a way of compensating for some other lack in functionality or flexibility of the protection controls.

INTERNAL KERNEL ARCHITECTURE

Much of the discussion up to this point has concerned the architectural place of a kernel in the larger system architecture, and the effect of kernel goals on that structure. Naturally, the internal structure of a kernel is also important. In general, the task of constructing a reliable kernel is little different from that of developing other highly reliable software. However, there are considerations peculiar to this application which deserve mention, including hardware selection, parallelism, and the effect of finite resource limits on attempts to use abstract type methods for organizing kernel code.

Hardware selection

Since hardware is the zero-th level in a structured, layered software architecture, unnecessary complexity in that interface is as undesirable as complexities in the specification of any other layer in the software, for the usual, good reasons. Therefore, hardware selection requires care, as illustrated earlier. It is worth noting that since the kernel software has already been minimized, the hardware impact on what remains can be substantial. In the UCLA kernel for example, approximately 40 percent of the code is device drivers, much of which could be eliminated if address relocation applied to device transfers.

Parallelism

While there may be some argument over whether algorithms that are expressed as parallel computations or strictly sequential code are inherently easier to understand (to some degree it certainly depends on the algorithm), it seems rather clear that certification or verification of sequential code is presently less difficult. That is the reason for specifying physical or logical uninterruptibility in the software architecture. This goal is, of course, one that has been followed at higher levels in systems since the early days of multiprogramming.

Abstract type structures

The concept of using extended types (such as illustrated by Simula classes,[18] CLU clusters,[19] Alphard forms,[7] or Euclid modules[20]) to organize software is quite valuable. However, Janson[15] points out that most efforts to employ strict type extension methods in programming are done with the underlying assumption of unlimited quantities of resources available. In the internal structure of an operating system or its kernel, this assumption is invalid, of course. One of the most obvious cases where finite resource limits interact with system structure occurs in virtual memory systems, potentially resulting in a structural circularity. Main memory is finite, so system software is constructed to give the illusion of a larger address space by sensing page faults and performing necessary I/Os in a manner invisible to the process involved. However, one may well wish to run this virtual memory software as a process like other processes. Further, the process management software can benefit from being written with the assumption that a large,

virtual address space is available. With the obvious circular dependency in mind, it is not clear whether process types or virtual memory types should be the lower level, with the other built on top. Reed[21] outlines a method of defining multiple levels of types to break the cycle. Lower levels implement a small fixed number or amount of the resource, using small amounts of resources in so doing. Higher levels reimplement the resource in large numbers or amounts, by multiplexing the lower level ones. In medium sized systems such as UCLA Unix, such a multiple tiered structure is not necessary.

CONFINEMENT

The problem of constructing a system so that an arbitrary program can be run in a fashion that assures the program cannot leak any of the information that it contains, is in practice unsolved. While clearly solvable in principle,[22–24] no actual solutions have been designed or implemented for real systems.

Importance

Some have argued instead that confinement is not a serious issue in practical applications. While this viewpoint is clearly reasonable when taken in the context of the current absence of reliable data security, several experiments with the Multics system have illustrated how real the problem may be. It was found possible in principle to drive a user terminal through interprocess communication performed by modulation of paging behavior in one process and the sensing of that modulation by the other. In a separate experiment, a similar bandwidth was discovered through use of the directory system. One process repeatedly made and deleted entries in a directory. This action changed the size of that directory, a value recorded in its parent directory. A second process read the size of that parent directory. In this particular test, the first process was not supposed to be able to communicate to the second.[11]

These two examples are instructive, for they illustrate a timing dependent and timing independent channel, respectively, both involving resource management. The bandwidth is enough to be of concern both to the military (one job might be a "Trojan horse" running with top secret classification, and the other running unclassified) as well as to industry (consider disgruntled programmers).

From a practical point of view, virtually all the channels mentioned by Lampson[23] or illustrated above are related to the allocation or management of resources, and the changes in system data that occur as a result of resource allocation decisions.

Storage and timing channels

Millen et al.[24] partition these channels into storage channels and timing channels. Storage channels are those which permit the passing of information through direct modification and reading of cells in the computer memory. Timing channels are those which permit one user to vary the real time rate at which another receives service, to a degree that the second user can sense. As the examples earlier show, the bandwidth of both types of channels can be significant.

It appears, however, that all resource storage channels can be changed into timing channels, and, on a large system, the bandwidth of the timing channels can be severely limited. Further, it appears that these actions can be taken without losing the values of multiprogramming, which is the primary cause of the problem. Resource storage channels can in general be changed to timing dependent channels in the following way. First, it appears that all direct reads and writes of system storage, such as that illustrated by the Multics file system problem, can be blocked by proper definitions of domains and the association of status data in the appropriate domains. What remains are resource allocation requests and status notifications. The use of resource availability as a channel can then be prevented merely by blocking the process requesting the resource until it is available. To be practical, this replacement of resource status replies by delays should be accompanied by the use of some form of a deadlock detection or prevention algorithm to insure that the system operates efficiently. That code need not necessarily be implemented in a security relevant way, as shown by the discussion of scheduling below.

Some believe that the bandwidth of timing channels can be limited by the scheduling characteristics that can be introduced in a large system. However, the existence of a timing channel is in principle more difficult to identify than a storage channel, since in the latter case there are explicit cells which are being accessed, even if interpretively through system code. While one might argue that if a process has only virtual time available to it, the timing channels are blocked, this viewpoint is probably not practical, given how many real time devices (therefore, clocks) actually exist on real systems. Nevertheless, once the bandwidth is limited to an amount comparable to that expected external to the computer system, the problem has in effect been solved.

Timing independent scheduling channels

While the preceding paragraphs may suggest a general approach to confinement, considerable care in the system architecture and implementation is required, as illustrated below. Earlier it was pointed out that functions such as the scheduling policy for resource allocation could be done outside a kernel. With respect to confinement, however, this approach creates difficulties. A reasonable scheduler must receive considerable information if it is to perform effective optimization. Since it can transmit information by ordering the completion of user requests, there is a clear potential for covert communication whether or not the scheduler had been written to cooperate.

Storage channels can in large degree be blocked however if the scheduler is designed as an information sink. This action can be taken if each user is permitted to submit only one request at a time, and wait for its completion before submitting another. Device completion status can be re-

turned via the kernel. Studies have shown that, for a reasonable number of jobs on a multiprogramming system, these restrictions are generally not ones with significant impacts on performance.[10]

However, such restrictions do not prevent collusion among user processes from providing timing independent channels. That is, a group of cooperating, legally communicating processes may be able to communicate with another such group, with which they are not permitted, even in the face of a scheduler designed to be an information sink. Whether or not this is possible depends on scheduler characteristics. Whenever it is possible for one process, by its own resource utilization actions, to affect the *relative order* in which members of other groups of processes are served, communication is possible.

That is, let processes A and B be one group, X and Y the other. Inter-group communication is not to be permitted. (They may be Ford and G.M. programs). Suppose that A (or B) increases its cpu to I/O ratio so much that the system's adaptive scheduler stops favoring cpu bound processes and begins favoring I/O bound jobs. X is cpu bound, Y is I/O bound. The order in which X and Y are run was changed by A or B. X and Y can easily determine the relative order in which they are run since they can legally communicate. Thus the A,B group has sent one bit to the X,Y group. Clearly this mechanism serves as a basis for extended bi-directional communication.

Many of the interesting scheduling algorithms that are used in practice have the characteristic outlined above, and in fact those that don't seem to ignore precisely that kind of information concerning user behavior upon which meaningful optimization is based. However, not all useful algorithms permit collusion. Round robin does not. The simple, limited algorithm implemented in the new Mitre Unix kernel, in which each process can only set its own priority, and the kernel merely runs that user process with the highest priority, also blocks collusion.

If confinement is felt to be important, considerable auditing of resource management mechanisms and policy will be necessary to block or satisfactorily limit the bandwidth of these kinds of channels.

CONCLUSION

We have attempted to distill the knowledge gained from various security kernel research efforts into a form that will be useful in guiding others who wish to develop highly secure, reliable system bases. It is also hoped that these perspectives are applicable to other environments than operating systems, especially such higher level software projects as message systems and data management, where privacy considerations are especially sensitive.

ACKNOWLEDGMENTS

This paper would not have been written had it not been for Clark Weissman, who often asked one of the authors how one designed a kernel. We also wish to thank Evelyn Walton, who largely built the UCLA kernel, and all the members of the UCLA Security Research Group who participated in discussions that helped form and refine these ideas.

REFERENCES

1. I.B.M. Corporation, *I.B.M. Virtual Machine Facility 370: Planning Guide,* Publication No. GC 20-1801-0, 1972.
2. Brinch Hansen, P., *Operating System Principles,* Prentice Hall 1973, 366 pp.
3. Lampson, B. W. and H. Sturgis, "Reflections on an Operating System Design," *Communications of the ACM,* 1976.
4. Schell, R., et al., "Preliminary Notes on the Design of Secure Military Computer Systems," USAF Technical Report ESD/MCI-73-1, January 1973.
5. Popek, G., "Protection Structures," *IEEE Computer,* June 1974, pp. 22-33.
6. Saltzer, J. H. and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE,* Vol. 63, No. 9, September 1975, pp. 1278-1306.
7. Wulf, W., et al., "HYDRA: The Kernel of a Multiprocessor Operating System," *Communications of the ACM,* Vol. 17, No. 6, June 1974, pp. 337-345.
8. Saltzer, J. H., "Protection and the Control of Information Sharing in Multics," *Communications of the ACM,* Vol. 17, No. 7, July 1974, pp. 388-402.
9. Ritchie, D. and K. Thompson, "The Unix Timesharing System," *Communications of the ACM,* Vol. 17, No. 7, July 1974, pp. 365-375.
10. Belady, L. and C. Weissman, "Experiments with Secure Resource Sharing for Virtual Machines," *Proceedings of IRIA International Workshop on Protection in Operating Systems,* Rocquencourt, France, August 13-14, 1974, pp. 27-34.
11. Saltzer, G., private communication, 1976.
12. Popek, G. and C. Kline, "A Verifiable Protection System," *Proceedings of the International Conference on Reliable Software,* May 1975, Los Angeles, California.
13. Gaines, R. S. and C. Sunshine, "A Secure NCP for Kernel Based Systems," RAND Internal memo, 1976.
14. Popek, G. and C. Kline, "Verifiable Secure Operating System Architectures," *Proceedings of 1974 NCC,* pp. 145-151.
15. Janson, P. A., "Removing the Dynamic Linker from the Security Kernel of a Computing Utility," MIT, Masters Thesis, June 1974, MAC TR-132, 128 pp.
16. Robinson, et al., "On Attaining Reliable Software for a Secure Operating System," *1975 International Conference on Reliable Software,* April 21-23, 1975, Los Angeles, California.
17. Kampe, M., C. Kline, G. Popek and E. Walton, "The UCLA Data Secure Unix Operating System," UCLA Technical Report, July 1977.
18. Dahl, O. and K. Nygaard, "SIMULA—An Algol-Based Simulation Language," *Communications of the ACM,* Vol. 9, No. 9, September 1966, pp. 671-82.
19. Liskov, B., "An Introduction to CLU," Computation Structures Group Memo 136, M.I.T., February 1976.
20. Popek, G., J. J. Horning, B. Lampson, J. Mitchell and R. London, "Notes on the Design of Euclid," *Proceedings of the Conference on Language Design for Reliable Software,* March 1977, Raleigh, North Carolina.
21. Reed, D. P., "Processor Multiplexing in a Layered Operating System," Masters Thesis, M.I.T., June 1976, 207 pp.
22. Fenton, J. S., "Memoryless Subsystems," *Computer Journal,* Vol. 17, No. 2, May 1974, pp. 143-147.
23. Lampson, B., "A Note on the Confinement Problem," *Communications of the ACM,* Vol. 16, No. 10, October 1973, pp. 613-615.
24. Millen, J. K., "Security Kernel Validation in Practice," *Communications of the ACM,* 1976.

Computer system security evaluation

by PETER G. NEUMANN
SRI International
Menlo Park, California

INTRODUCTION

This paper considers the problem of attaining computer systems and applications programs that are both highly secure and highly reliable. It contrasts two current alternative approaches, one remedial, the other preventive. A remedial approach is outlined based on a classification of software security violations suggested by Bisbey, Carlstedt, and Hollingworth at ISI. This remedial analysis is then related to a preventive approach, illustrated here by the formal SRI Hierarchical Development Methodology. Evaluation of system security is then considered by combining concepts from the preventive and remedial approaches. This combination of techniques seems to have significant potential in the attainment and evaluation of computer system security. Illustrations are given for three types of systems, the first two being systems explicitly designed with security in mind, and the first of those being designed according to a formal methodology. The first system is the SRI design for a Provably Secure Operating System (PSOS), the second is Multics, and the third is UNIX. (The reader familiar with security may wish to skim the next two sections.)

BACKGROUND

Computer systems and applications are increasingly being called on to provide reliable security, although most existing commercial computer systems are incapable of supporting various security-critical applications. Until recently, the task of obtaining highly secure systems has typically been considered to be very difficult, and avoided on the grounds that good solutions would be very expensive. There has been little deep understanding of either how to develop secure systems and applications, or how to assess security; however, there have been efforts to detect insecurity. The interactions between security on the one hand and reliability and survivability on the other have also been largely ignored, as have the interactions with dynamic auditing of security. The term "system defensiveness" is used here to imply security, reliability, survivability, and the auditability of any events that might compromise these aspects of computer behavior.

Recently, the situation has been improving (e.g., see the survey in Shankar[1]). Two approaches to attaining better security are considered here, one basically remedial, the other basically preventive. The first approach involves assessing the security of computer systems (particularly existing ones) and attempting to patch around any security flaws thereby uncovered. A variety of techniques have been used to detect flaws that reduce security. These include searching code for certain patterns of program statements corresponding to classes of would-be security flaws (e.g., Bisbey[2]), as well as more traditional experimental penetration attempts (e.g., McPhee[3]). The second approach involves designing new systems that are intrinsically secure and whose security can in some way be convincingly established. This approach might rely on a suitable design methodology (e.g., Robinson et al.[4]), combining formal specifications (e.g., Parnas,[5] Robinson et al.[4]), suitable design structures, the use of suitable programming languages (e.g., see Löhr[6]), and supporting tools. Examples of such approaches that also support formal verification are found in Robinson et al.[4] and Good.[7]

With respect to the remedial approach, experience in attempting to penetrate allegedly secure systems leads to several observations. First, penetrators do not generally have to work very hard to find major security flaws in traditionally developed systems. Second, patches that attempt to remove such flaws are themselves often flawed. Third, this approach is intrinsically limited because attempts to characterize the still undetected security flaws are speculative, at best. In general, attempting to retrofit security into a basically insecure system is of limited effectiveness, and inevitably leaves much in doubt.

In the long run, use of the preventive approach is likely to be significantly more productive. This approach should proceed with security as a fundamental design goal from the outset. It may include explicit statements of requirements and formal specifications of the design. It should make use of a modern programming language, and might employ formal proofs of significant properties regarding system behavior, such as security (e.g., see Neumann et al.[8]). On the basis of recent research, it is now possible to assess the security of a precisely specified design, prior to implementation, and then subsequently to assess the security provided by the implementation.

The conclusions of Glaseman et al.[9] are extremely pessimistic, at least for the near future, with respect to obtaining quantitative assessments of the cost (risk) associated with various security violations. Similar conclusions are drawn

in the present writing, with regard to attaining meaningful measures of security in conventionally designed systems. However, there are some hopes for attaining secure systems. First, it is helpful to assess insecurity—although it is not very reassuring to know just that another security flaw has been found. Second, it is possible to consider relative measures of insecurity for different classes of would-be violators such as skilled system programmers, data processing supervisors, system operators, skilled users, and casual users. Some security flaws may provide threats only from users in certain of these classes. Knowledge of the would-be violators and the degree to which they are trusted is thus fundamental to any evaluation of risk. Third, there is some short-term hope in a combination of the remedial and preventive approaches in redesigning an existing system so that its security-relevant portions can be isolated into a small and cleanly defined collection of system programs (a "security kernel" plus a few trusted processes, jointly responsible for the security of the system). However, the effectiveness of this combination can be reduced by restrictions imposed on the required redesign, such as the need to maintain compatibility with the original (insecure) system interface. It also may confuse policy (e.g., security strategies) with mechanism (e.g., the protection mechanisms upon which the policy is implemented), potentially embedding high-level policy issues inflexibly into low-level mechanisms. In any case, it is felt that the preventive approach must be involved whenever new systems are to provide substantially more security than existing systems. In particular, the use of a formal development methodology for design and implementation can provide a powerful basis for intrinsic and quantitatively assessable security.

SYSTEM DEFENSES

Defensiveness is considered here as a generalized notion of security, and implies the nonexistence of inappropriate behavior—insofar as possible. For present purposes, defensiveness consists of related and partially overlapping components, namely security, reliability, availability (including recoverability), and auditability.

(1) *Security* as used here involves the protection of the system, its applications, and the shared resources from misuse. The context considered is purposely a broad one. It includes the notions of preventing unauthorized acquisition and unauthorized modification of information, that is, assuring "confidentiality" and "integrity," respectively. It is intended to cover both system security and user data security. [In usage by MITRE, viz., Biba[10] "(data) integrity" is a precise formal dual of multilevel "(data) security" (i.e., confidentiality), the former referring to the writing of data, the latter to the reading of data.] In usage here, security also includes the prevention of denial of service (maintaining the integrity of resources) and the prevention (where possible, or otherwise the limitation) of information leakage through clandestine channels (such as signalling through shared resources or through shared timing channels). (See Lampson,[11] Lipner,[12] Neumann et al.[8]) In certain applications the notion of security may refer to or include multilevel (e.g., military) security with levels such as TOP SECRET, SECRET, etc. (See Bell and LaPadula,[13] Feiertag et al.[14])

(2) *Reliability, Availability,* and *Recovery* involve (respectively) the correctness of system security (including data integrity) and other system functions; the maintenance of a secure running system; and the restoration of a secure running system following any error, accident, willful damage, or disaster that caused a loss of service or security.

(3) *Auditability* involves the monitoring of the continued existence of system security and reliability, including the detection of anomalous or potentially threatening behavior.

Continuous maintenance of security depends on appropriate system hardware and software, including concepts of fault-tolerant architecture and reliable software. It also depends on an appropriate operational environment. It requires global planning and management. The implications of an unreliable system on security are particularly insidious, and thus must be considered along with the more typical considerations of security.

It is a difficult task to assure dynamically the continuous absence of all would-be security violations, e.g., to audit misuse of an intrinsically insecure system. It seems much more fruitful to work on the design of better systems, and then to develop auditing procedures as an integral part of the design. (For example, auditing should obey security policies wherever possible.) However, once a breach of system defenses is discovered, it should be fixed as rapidly as possible. In the sense that the effects of a security violation may already have leaked out, it may not be possible to provide dynamic recovery in the strict sense. Thus auditing is important to help limit the propagation. Good design should anticipate the needs of both auditing and recovery, with auditing that strongly limits the propagation effects of detected violations and that simplifies recovery therefrom.

Given an existing system not designed to be secure, the elimination of potential threats becomes more and more difficult after the first few threats are removed. The process is time-consuming, frustrating, unpredictable, and ultimately limited by the knowledge that although fewer threats are being discovered, many more may still remain undiscovered. It is particularly perverse that the changes necessary to remove discovered threats may themselves lead to new violations, particularly in conventional systems.

Given an arbitrary system, it would be useful to have some assessment of the penetrability of the system. However, even the most innocent-looking flaw may in fact be a hole in the dike that leads to total inundation. In general, the risks from system violations can best be avoided by a combination of good design, good implementation, good operations, and good management, all done with defensiveness in mind. A combination of testing and penetration studies,

and possibly some formal verification is recommended, commensurate with the desired system requirements, the expected difficulty of penetration, and the cost of having violations. Formal verification for the most security-critical portions of the design and its implementation is expected to be cost-effective in the near future. Recent techniques for verifying design properties, independent of any implementations, are particularly promising.

The remainder of this paper is organized as follows. First, a categorization of various typical system violations is given, based on the work of Bisbey, Carlstedt, and Hollingworth at ISI. This provides the framework for a remedial analysis of any particular system. A collection of preventive criteria is then given whose presence in the design, implementation, evolution, and operation of new computer systems may result in systems that are far more defensive than the conventional systems available today.

It is felt that these categories of violations and criteria for defensiveness should be very useful for evaluating the development of new systems and applications, as well as existing ones. They are used here first for a consideration of the design of PSOS (a Provably Secure Operating System), intended to be demonstrably secure. Multics and UNIX are then considered as two (extreme) examples of existing systems. It is seen that a system developed according to the preventive criteria intrinsically avoids many of the flaws indicated by the remedial approach.

VIOLATIONS OF SYSTEM DEFENSES

Nielsen et al.[15] have analyzed over 300 cases of computer misuse (from the files of Donn Parker) and have identified seven types of violations to system defenses, including cases of disaster, accidents, hoaxes, threats and extortions. (Defensiveness is therein called "integrity.") However, system violations are involved in the preponderance of cases (e.g., undesired acquisition, modification, insertion, or destruction of data, along with various other forms of system penetration). Some of the cases involve undesired denial of service. The sophistication of leakage through clandestine channels was apparently unnecessary, given the ease of penetration by simpler means. Nevertheless, in deference to Murphy's Law ("If it can happen, it will"), all plausible violations should be anticipated, and covered by various safeguards whenever the threats are deemed significant.

Classes of system penetrations

Bisbey, Carlstedt, and Hollingworth at the USC Information Sciences Institute (ISI) have studied techniques for identifying potentially vulnerable sections of code, including some techniques that could be carried out automatically. In so doing, they identified 10 categories of system flaws, each capable of producing security violations.[2] These categories have evolved over the last few years, with each violation observed at ISI fitting into at least one of these categories. However, there is no pretense that these categories are either canonical or complete. (Although they were conceived primarily to be applicable to software, most of them are in fact also applicable to hardware.)

Two of the 10 ISI categories are closely related and have been lumped together here. For present purposes, the resulting nine topical categories are grouped into four generic groups of flaws. Each has both design and implementation aspects.

(A) improper protection (initialization and enforcement);
(B) improper validation;
(C) improper synchronization;
(D) improper choice of operation or operand.

The nine topical categories can be associated with these groups as follows.

PROTECTION (initialization and enforcement):
(1) improper choice of initial protection domain;
(2) improper isolation of implementation detail;
(3) improper change (e.g., a value or condition changing between its time of validation and its time of use);
(4) improper naming;
(5) improper (incomplete) deallocation or deletion;
VALIDATION:
(6) improper validation;
SEQUENCING:
(7) improper indivisibility;
(8) improper sequencing;
OPERATION CHOICE:
(9) improper operation or operand selection.

These categories are illustrated in Table I. They cover many different security violations, including undesired reading, writing, and deleting, undesired denial of service, and to some extent undesired leakage through clandestine information channels. For example, visibility of implementation detail (category 2) often leads to potential leakage channels. However, leakage channels also exist in systems that are otherwise secure, because of the natural visibility of elapsed time.

It is not claimed that the four generic categories are orthogonal. Nevertheless, they provide very useful typical cases. Within the first generic category ("improper protection"), the five topical categories exhibit considerable overlap. (Category 1 deals with initialization, while categories 2-5 provide paradigms of improper enforcement.) In fact, there exists a sequence of flaws that exhibits an ordering of (1)⇒(2)⇒(3)⇒(4)⇒(5), as follows. An incorrect choice of protection partition (1) can result in the inability to isolate the implementation of an abstraction from the use of that abstraction (2). Such inability can result in a value presented at the time of call (and expected subsequently to remain constant) being changed during execution (3). Such a flaw can result in a naming problem (4), in which two different paths to the (apparently) same data can give different results. Finally, a naming problem may create a residue problem (5), e.g., if a local symbolic name is not unbound when the object is deleted with reference to its global name—or vice versa. (Since each category may reappear at different levels

TABLE I.—Categories of Protection Flaws (and examples) (Based on Bisbey, Carlstedt, Hollingworth at ISI)

1. Incorrect choice of protection domain or security partition
   (a security-critical function manipulating critical data directly accessible to the user; incorrect initial assignment of security or integrity level at system generation, configuration, or initialization)
2. Exposed representations or implementation detail
   (bypassing an abstraction, e.g., direct manipulation of a hidden data structure such as unmediated user modification of a directory entry; user use of an absolute I-O address; note that the visibility of timing information provides a generic leakage channel, e.g., drawing inferences from page fault activity)
3. Inconsistency of data over time
   (noninvariance of parameters, e.g., change in value of a parameter in a call by reference; change in a file accessible to different processes, e.g., in an improperly protected, shared process directory)
4. Naming problems
   (aliasing, e.g., two distinct names for the same object not being treated identically; ambiguity resulting from use of the same local name for two distinct objects [which may also involve a residue problem, e.g., if name persists in some system table])
5. Residues in allocation and deallocation
   (incomplete deletion, revocation, or deallocation, e.g., such that an apparently deleted value is still accessible—in core, disk, archive store, etc.; incomplete cleanup on abort; ignoring terminal hangup).
6. Nonvalidation of critical conditions and operands
   (invalid or unconstrained parameters such as an out-of-bounds virtual address or absolute I-O address; lack of strong type checking, e.g., a pointer to a structure of the wrong type; absence of quota limit stops such as bounds on queue sizes or number of processes, with overflows resulting in possible system or user crashes)
7. Indivisibility problems (in multiprogramming)
   (interrupted atomic operations, e.g., incomplete interrupt handling [quit during login resulting in partial success, or in perpetual lockup of interlocked data]; faulty read-alter-rewrite in hardware)
8. Serialization problems (in multiprogramming, multiprocessing)
   (incorrect sequencing [e.g., wrong order], improper isolation of atomic operations from one another [e.g., reading during writing, or concurrency among different directory commands on the same directory]; critical race conditions in implementation; deadlocks and deadly embraces)
9. Incorrect choice of operation or operand
   (use of the wrong function, producing incorrect results; use of an unfair scheduling algorithm, producing correct results for each scheduled process, but denying service completely to certain users)

in a hierarchical design, other such orderings may also exist.) Thus the most primitive descriptor of these protection flaws is something like "improper domain selection and enforcement."

There is some overlap between the two topical categories of sequencing flaws. The indivisibility and serialization categories, (7) and (8), respectively, are related when considered at different levels of abstraction, since (logically) atomic operations appearing at one level may in fact be implemented out of (logically) atomic operations at a lower level, with appropriate interlocks. That is, a serialization problem at one level may apparently manifest itself as an indivisibility problem at the next higher level.

Validation problems usually involve the omission of a check on argument validity or on quota limits. The inconsistency of data over time (category 3 above) might conceivably be viewed as a validation problem. (This is the time-of-check-to-time-of-use flaw, or "TOCTTOU," considered by McPhee, Weissman, and others.) However, this view is somewhat like locking the barn door after the horse has escaped: the primitive flaw is not a lack of validation, but rather the lack of either protection or interlocks that permitted the value to change in the first place.

Categories 3, 5, and 6 above have been the subject of experimental study at ISI, including the development of computer tools that search given programs for specific types of violations—notably interprocedure data dependencies, potential inconsistencies of data values over time, and potential residues. (See Bisbey et al.,[16,17] Carlstedt et al.,[18] Carlstedt,[19] and Hollingworth and Bisbey.[20]) However, the analysis of most categories is not generally amenable to systematic investigation.

It is useful here to identify some of the symptoms that may underly flaws of each particular category. Such an identification is attempted in Table II, in which are listed various conditions that can (but not always do) cause flaws in security.

DEFENSIVE SYSTEMS

In general, good design of the hardware and software and good implementation are highly important. In existing sys-

TABLE II.—Symptoms of Potential Protection Flaws, by Category

1. Domain choice. All programs or human actions relating to the initialization or interpretation of protection information are suspect, e.g., any setting or changing of a security level, particularly any action that lessens security (e.g., downgrading).
2. Exposed representations. Any direct visibility or use of implementation detail is suspect. Any use of absolute addresses for memory or input-output. Nonvirtual resources. Direct access to a data structure that is normally used as an abstract data object. Serial dependence within logically combinational functions.
3. Data inconsistency. A called procedure fetches the value of a parameter more than once, or fetches a value it just stored. A parameter is passed by name or by reference. (The value may change between call and return.) An output value is overlaid on top of an input value. Reference is made to a value that is self-modifying upon being accessed. (See Bisbey et al.,[16] Carlstedt et al.[18])
4. Naming. Any object for which two different names can exist is suspect. Use of a local name in one context, a global name in another, e.g., a virtual and a nonvirtual name. Any use of a table index where protection is expected.
5. Residues. Physical deletion of contents is suspect whenever deferred beyond logical deletion, e.g., deferred until reuse of media space. Readable free pools. Accessible backup storage. Reuse of an index or slot number after deletion of entry. (See Hollingworth and Bisbey,[20] which enables isolation of all allocations and deallocations, and searches for potential residues.)
6. Nonvalidation. The absence of any checks on protection information upon access to sharable data is usually indicative of a flaw. The absence of any checks on an input variable or parameter, on its type or value range, or even on the existence of data is suspect. Lack of quotas on real resources or on different virtual partitions of resource usage can provide a leakage channel across partitions. Validation of status at the time of a request for which status may change, without revalidation on completion. (See Carlstedt,[19] Bisbey et al.[17].)
7. Indivisibility. Any allegedly noninterruptible or indivisible operation is suspect, as is the mechanism for achieving indivisibility.
8. Serialization. Any overlapping of operations using the same data is suspect, either with different uses of the same operation, or simultaneous uses of different operations on the same data base.
9. Choice of operation or operand. This category is very hard to formalize. Potentially any operation can be improperly chosen. Operations and data types that do not correspond are suspect, as is the use of mismatched type declarations.

tems, it is usually bad design and bad implementation that lead to violations of system defenses. Thus, good design and good implementation should themselves be considered as (meta-)safeguards relating to software. The *a priori* use of good design practices and good implementation practices is in general enormously more effective than the *a posteriori* retrofitting of patches that attempt to strengthen the defenses.

Factors influencing development of defensive systems

The factors summarized in Table III are relevant to improving the defensiveness of a design and its implementation. Many of these factors have profound effects throughout the development process, involving requirements specifications, design, implementation, system maintenance, long-term evolution, and operation.

TABLE III.—Factors Influencing Defensiveness in Systems and Applications

Well-defined and well-understood requirements, established clearly and agreed upon in advance
Good design (e.g., modularly structured, especially hierarchically, with strict isolation of application programs and system programs, strongly typed operations, unified treatment of storage, input-output [e.g., mapped virtual access])
Suitable implementation languages (e.g., strong typing, avoidance of aliasing, constrained argument passing [such as use of call by value where data inconsistency may be a problem], hiding of implementation detail and device dependence wherever possible, clean control structures, encapsulation of data types)
Well-defined and understandable specifications for the system hardware and software
Structured implementation, reflecting the modularity of the design wherever appropriate, and structured initialization (e.g., hierarchical)
Systematic handling of exception conditions and quota limits
Auditing and recovery integrated into system design, e.g., hierarchical
Careful debugging, testing, verification
Good management of system development (e.g., respecting these factors)
Lessening the need for management as a result of simplifications resulting from use of these factors
Good management of system operation (e.g., rigid adherence to system generation and evolution protocols)
Nonreliance on secrecy of design and implementation
Awareness of the user community (e.g., enforcing the use of random pronounceable passwords rather than guessable ones)

Use of a formal development methodology for secure system development

As an indication of how systems can fare with respect to the presence or absence of potential security flaws, a system is considered here that has been conceived under rather unusual—and hopefully optimal—conditions. This system is PSOS, a Provably Secure Operating System.[8] PSOS has been designed from the outset to be able to support advanced security requirements beyond those of existing systems, and has taken advantage of essentially all of the above factors influencing good design and implementation. In particular, it has been designed according to formally stated requirements, has been formally specified, and has been subjected to formal proofs of its critical design properties. It is designed according to the SRI Hierarchical Design Methodology (HDM), described in Robinson et al.,[4] Neumann et al.,[8] and Robinson and Levitt.[21] The design is represented in a formal language, SPECIAL (A SPECIfication and Assertion Language, Roubine and Robinson.[22]). The system has not yet been implemented, but the nature of would-be implementations has been characterized. The design has been oriented toward provability of the design and of subsequent implementations. (HDM is also being used in the design and implementation of the SIFT computer system for an ultrareliable commercial aircraft application [for which proofs of elementary fault-tolerance properties have been undertaken], the design of a real-time operating system, the design of a family of related message processing systems, and the development of provable security kernels.)

PSOS is a capability-based system in which each capability acts as a protected name or token for an object. A capability for an object is protected, in that its creation and its transfer to other users or processes can be controlled, and once created, it can never be altered. It protects the object to which it refers, in that its presentation is required for that object to be accessed, including access rights appropriate to the operation being performed. (Certain PSOS applications take advantage of store-limited capabilities, which cannot be transferred out of the containing process.)

If formal verification of the design or its implementation is desired, then the following also contribute, both separately and collectively:

Formally stated requirements
Formally specified design, including specifications of modules and their interrelationships (e.g., data representations)
Formal proofs of correspondence between design specifications and requirements
Formal axiomatization of the programming language
Formal proofs of consistency of programs with design specifications
Formal axiomatization of the hardware/microcode
Formal proofs of consistency of hardware/microcode with hardware specifications

An entity analogous to a capability exists in SPECIAL, called a *designator*. A designator serves as a protected name of an object. Its uniqueness in SPECIAL is part of the specification language. In the implementation of PSOS, this uniqueness can be easily guaranteed by the use of capabilities. (Other solutions exist in other systems, such as descriptors.)

Table IV gives an indication of how the use of the methodology together with the use of a suitable programming language can overcome each of the nine categories of flaws summarized in Table II. It is seen that the use of SPECIAL has a very significant impact on the avoidance of these flaws in design. Most of the comments on the design specification in the table are generally applicable to any system specified in SPECIAL. Similarly, the use of the hierarchical design methodology has significant impact on the avoidance of these flaws in implementation.

The use of a suitable modern programming language can also contribute considerably. A language such as Euclid,[23] Modula,[24] Texas' Gypsy,[7] or SRI's HDM-compatible ILPL

TABLE IV.—The Influence of the SRI Methodology and Programming Languages (PL) on Avoiding Characteristic Flaws, both in General and in PSOS (P:)

Category of Flaw	Design (in SPECIAL)	Implementation
1. Domain choice	Design proofs.	Program proofs.
2. Exposed representations	Hidden by hierarchical levels of abstraction. P: Extended types further aid hiding.	PL encapsulation; proofs. P: Extended type mechanism.
3. Data inconsistency	None. No overwrite. Conceptual call by value only.	PL call by value; proofs. P: Argument values on stack unalterable.
4. Naming	Unique designators, nonbypassable, only means of naming. P: Capabilities unique.	PL no aliasing; proofs. P: Uniqueness and nonbypassability of capabilities aided by tagging.
5. Residues	None in specifications. Deallocated values become UNDEFINED, are thereafter unnamable. P: Residues protected.	PL dynamic object deletion and creation; proofs. P: nonreusable identifiers; contents zeroed if desirable on deallocation.
6. Argument and critical condition nonvalidation	Avoidable. Strong type checking, explicit subtypes, explicit exception and error conditions.	PL strong typing; compile or run-time checking. P: Extended types; I-O addresses mapped (via capabilities).
7. Indivisibility	Specifications for each function are logically indivisible.	PL synch primitives; proofs. Completion or no-effect exception provable.
8. Serialization	Sequencing invisible in specifications.	Benignness of overlap provable.
9. Operation choice	Faulty specifications detected by spec proofs. Type checking also helps.	Program proofs.

Many of the characteristic flaws can be avoided in design by the use of a methodology such as the SRI methodology, and in implementation by the use of a suitable programming language.
Note: "PL" refers to language features found in such languages as Euclid, ILPL, and the emerging DoD/1 languages. "P:" refers to PSOS. (See text.)

(due to Larry Robinson, and documented in Neumann et al.[8]) would be particularly helpful, as might the languages emerging from the DoD/1 effort. (ILPL is an extremely basic intermediate-level programming language that gains its simplicity from the power of the supporting methodology, whose formal specifications provide [for example] its data structures as a by-product of the hierarchical levels of abstraction.) Some of the major contributions that such languages can make are indicated generically by the "PL" entries in Table IV. In most cases these contributions result from intrinsic properties of the language. In other cases they may result from simply enforceable language restrictions. Note that not all of the above languages have all of the desired facilities. (At present, Euclid and ILPL [the latter taken together with HDM] are probably the most complete in this respect. Modula and Gypsy are currently being extended, and support for Gypsy and ILPL is currently being developed.)

Finally, for many of these flaws, relatively simple proofs are possible to demonstrate that the particular flaw is absent in various explicit forms. The design aspects result from properties of the specifications, e.g., they follow from proofs of consistency between the specifications and formal requirements, or are intrinsic to the methodology and the specification language. The implementation aspects result from proofs of program correctness, or are intrinsic to the use of the programming language. With this methodology, it is sufficient to show consistency between the programs and their specifications.[21] However, many proofs are possible without having to prove program correctness in general. Nevertheless, program proofs are becoming feasible as tools to support them continue to be developed.

The comments in Table IV are generically applicable to systems designed and implemented according to the methodology. In addition, those prefixed with a "P:" are particularized to the design and the would-be implementation of PSOS (e.g., in one of the above-mentioned languages). It is seen that a system designed according to a formal methodology like HDM is likely to avoid the characteristic flaws without much additional effort. In particular, PSOS has none of these flaws intrinsic to its design, and it is believed that an implementation essentially free of these flaws can be developed—and if desired, proved.

Above and beyond merely avoiding the nine characteristic problems, the use of HDM has a significant impact on the integrity of PSOS in other ways. These include the explicit statement of formal requirements for security, formal specifications for the design and its hierarchical structure, a unified design, a systematic approach to the handling of exception conditions and quota limits, and a structured implementation capable of taking advantage of the hierarchical design without losing much in efficiency. These factors contribute to the suitability of the design and implementation, and to the possibility of carrying out formal proofs, if desired. This additional impact is summarized in Table V.

Application of this evaluative approach to other systems

The evaluation of PSOS with respect to security flaws is clearly favorable. This is not surprising, in that PSOS was designed with security in mind from the outset, using the formal SRI methodology (which itself evolved simultaneously as a result of its application to PSOS). However, it is instructive to consider two other types of systems: the first which designed with security in mind, but without a formal methodology; the second designed with security only superficially in mind for its implementation. The examples taken are Multics and UNIX. (Conventional commercial operating systems are ignored here, as they are for the most

TABLE V.—Additional Influences of the SRI Methodology on the Integrity of PSOS

Formal requirements of security: basic principles concerning acquisition and modification of information, and a formal model for multilevel (military) security. These provide simple intuitively understandable design objectives.

Formal specifications of the design. These provide precise specifications for the system. Their consistency with the formal requirements can be formally proved. They provide the basis for implementation, as well as the basis for proofs of consistency of the implementation and the design. Proven specifications also provide a strong basis for compatible alternate implementations on the same or different hardware.

Hierarchical structure of the design. This encourages separation of policy and mechanism, and simplifies the initial implementation significantly. Recovery is done hierarchically, as is initialization. Recovery is therefore predictable and controllable. This increases the persistence of security during failure (partial or total).

Unified design. Auditing functions conform to the security requirements. Recovery functions are also integrated into the design and use standard functions. Hardware design is also integrated with software design. This increases the ability of the hardware to provide an appropriate basis for the software. [Use of a pronounceable password generator might be enforced to avoid guessable passwords.]

Systematic handling of exception conditions and quota limits. This increases the predictability of system behavior, and permits proofs of completion. Most information channels resulting from missing exception conditions in specifications can be detected and eliminated.

Implementation. To simplify the maintenance of security during system evolution, a highly compartmentalized implementation is very helpful. The hierarchical structure of the design can be retained in implementation wherever it is efficient to do so, and can otherwise be simplified without compromising the security of the system. The choice of implementation language would be constrained to support data abstraction, strong typing, exception conditions, etc. See Table IV.

Verifiability. Having followed the design methodology of using formal specifications for the user-visible functions, it is relatively simple to prove properties of the PSOS design or of an application environment, e.g., supporting multilevel security. Similarly, with the hierarchical design and formal specs for internal functions, program proving seems feasible, assuming an appropriate choice of programming language. The language features noted above all tend to improve verifiability.

part intrinsically insecure.) In essence, Multics is a large-scale general-purpose system with security superior to other commercial systems, with some emphasis on its being able to recover from outage, and some emphasis on audit. UNIX, on the other hand, makes little pretense of being secure, except for its attempts to provide access control and its encryption of the password file. It is fundamentally insecure.

Multics,[25] like PSOS, is a system that was innovatively designed (albeit in 1965) to take advantage of what was then known about advanced design and implementation techniques. UNIX[26] is a system that was developed at Bell Labs to take advantage of then recent operating system experience (e.g., Multics), on a much smaller scale—with judicious choice of simplifications intended to give high efficiency. However, UNIX was designed to be a system operating in a cooperative user environment, and was not conceived as a system providing any rugged sense of security. Nevertheless, it is a widely used and highly useful system. The discussion here is not intended in any way as a condemnation of UNIX as an insecure system, but rather as a simple illustration of how insecure a system can be in the absence of a pervasive concern for security from the outset.

Table VI presents a summary of how Multics and UNIX fare with respect to the nine characteristic problems. It is clear from the table that Multics is relatively secure. It is equally clear that UNIX is not. Table VII presents a comparison in terms of the methodological concepts of Table V.

After considerable improvement upon the implementation of a basically sound design, Multics has reached a level of significant security, and has become difficult to penetrate. Although it was subjected to various penetration efforts on its earlier hardware, and was indeed penetrated, the current hardware and software have long since removed the flaws permitting those penetrations.

UNIX has thus far apparently been used only in benign

TABLE VI.—Evaluation of Multics and UNIX with Respect to Characteristic Flaws

Category of Flaw	Multics	UNIX
1. Domain choice	Good. System, users in multiple rings. Ring 0 capture omnipotent, but unlikely. Outward migration of less critical functions. Provable layered security kernel designed by Honeywell, but not implemented.	Poor. Nonstratified. Capture of the entire system easy through superuser infiltration. Security kernel implementations exist (UCLA, MITRE), improving on original decomposition.
2. Exposed representation	Access within a ring all or nothing. Considerable hiding via ring mechanism.	Superuser easy to capture, defaults open. Trojan horses galore. Process temporaries readable and writable, main memory readable.
3. Data inconsistency	Argument pointer is copied onto stack by call. Arguments are copied within the system code to avoid this flaw internally.	Argument itself is copied, but only by convention. Process-temporary files are alterable in midcomputation by user or other users.
4. Naming	Collisions of local names: Trojan horse!	Default on directory entries: unprotected. Memory aliases (''/dev/mem''). Trojan horses abound.
5. Residues	Core zeroed only before reallocation, but is not virtually addressable after deallocation. Disk never zeroed, just overwritten. Residues after crash.	Core zeroed only before reallocation, and is until then still readable. Disk never zeroed, just overwritten. Residues after crash.
6. Nonvalidation	Hardware checking at ring and segment levels. All kernel args validated. Compile-time data-type checking.	Very few checks on resource or I-O bounds, interuser ops. Easy to crash.
7. Indivisibility	Locking makes kernel ops atomic. Exception handlers force no-effect noncompletion.	Exception handling bad.
8. Serialization	Locks prevent overlap. Lock hierarchy used to avoid deadlocks.	OK? System functions logically synchronous (until IPC installed).
9. Op choice	Possible problems?	Possible problems?

TABLE VII.—Evaluation of Multics and UNIX with Respect to Methodological Considerations

Influence	Multics	UNIX
Requirements	Informal	Informal
Specifications	Implementation laden	Implementation laden
Design structure	Implicit hierarchy	Procedural structure
Design	Fairly unified	Fairly clean
Exceptions,quotas	Fairly systematic	Poor
Implementation	Segmentation useful	Device independence OK
Programming language	PL/1 somewhat more helpful than C	C poor on abstraction, strong typing
Verifiability	Difficult because of size, PL/I, no specs	Pointless because of insecurity, C, no specs

environments, and is at present simple to penetrate. It appears that major renovation would be required to substantially improve its security (although a planned new release of UNIX is expected to eliminate a few of the problems).

From the methodological considerations of Table VII, Multics again appears to be better off than UNIX. However, from this vantage point, UNIX does not seem to be irreparably insecure. For example, a reimplementation with the imposition of constraints on C (or modifications to the language) and better handling of exceptions would be beneficial.

It is interesting to note that each of these three systems has been considered as a basis for supporting multilevel security. The PSOS design is augmented with a multilevel security policy manager that can be efficiently implemented on top of PSOS (or within it, if that is to be the only policy).[14] Multics and UNIX both have experimental redesigns retrofitting a kernel responsible for system security into the existing design. MIT, Honeywell, and the Air Force have undertaken the restructuring of the existing Multics system to improve (among other things) its security. As a component of that work, Honeywell has designed a Multics security kernel, for which formal requirements and formal specifications exist (using the SRI methodology). However, the funding for that project (Guardian) has expired before the kernel could be implemented. UCLA[27] and MITRE have independently designed and implemented prototype security kernels to be retrofitted into UNIX. These prototypes have provided the impetus for a competitive design effort between two groups (FORD-Aerospace and SRI as one group, TRW as the other), expected to result in a demonstrably secure kernel upon which is built a secure version of UNIX.

CONCLUSIONS

This document attempts to take a broad view of the development of secure systems, combining a remedial approach with a preventive approach. In particular, it characterizes various common flaws that can lead to security violations. It then uses this characterization to evaluate a constructive methodological approach to computer system development, and shows how such a methodology can intrinsically tend to avoid the characteristic flaws. This combination is thought to be useful for analyzing existing systems and for developing new systems.

Given a computer system whose design did not originally have security as a major goal, and which consequently may be flawed, it is in general extremely difficult to converge on a secure system by successively patching flaws as they are recognized. Nevertheless, the search for such flaws is desirable. In general, given a system that has evolved after extensive penetrate-and-patch efforts, or given a computer system that has been expressly designed to be secure, it is still a difficult matter to assess how secure the system really is. The evaluative approach presented here is felt to be useful. However, compatibly with this approach, recent advances in formal proof seem increasingly applicable. Such advances indicate the feasibility of proofs of the security of a design, given a suitable formal representation of the design and formal statements of security requirements. These proofs may be carried out prior to and independent of any particular implementation of that design. In addition, proofs of selected critical properties are feasible. Furthermore, the technology for handling proofs of consistency between design specifications and programs implementing those specifications is progressing well. Various semiautomatic computer tools for carrying out such proofs now exist and are being integrated. However, the approach outlined here is aimed at improving the resulting system dramatically, even if no program proofs are ever carried through.

A goal that emerges here is to be able to gain increasing confidence in the design and implementation throughout a system development, and to minimize reimplementation late in the development process by avoiding fundamental design flaws early in the process. The combination of approaches discussed here is seen to be relevant. A highly secure system can be attained, with a well-conceived design, careful implementation, selected formal proofs (particularly proofs of design properties), and the use of penetration efforts. However, it should be emphasized that a completely secure system is unlikely in any case, partly because of the intrinsic problems presented by leakage through timing channels.

PSOS is an example of a new system design whose stated goals from the beginning included support for advanced security requirements and provability. The experience gained in that and related efforts is used here to establish criteria that appear to be useful in helping to evaluate other efforts. For example, the ISI criteria and the methodological considerations clearly show some of the strengths of PSOS and Multics. It is hoped that this paper will lead to further refinements in security evaluation, and that it will be useful in evaluating other systems. It is also hoped that it will influence the subsequent development of secure computer systems.

ACKNOWLEDGMENTS

The writing of this paper has been supported by National Science Foundation Grant No. DCR 74-23774. The author is grateful to Richard Bisbey, Jim Carlstedt, and Dennis Hollingworth for sharing some of their experience, and to

Norman R. Nielsen and Brian Ruder for their helpful comments. The central person responsible for the SRI methodology (HDM) discussed here is Larry Robinson. He, as well as Rich Feiertag, Karl Levitt and Bob Boyer have made major contributions to the design of PSOS and to proofs of security. The author is indebted to the reader who has worked his way through the varied levels of discourse of this paper, from introductory verbosity to tabular terseness, recognizing that this paper may not have succeeded in trying to be meaningful to a wide range of readers.

REFERENCES

1. Shankar, K. S., "The Total Computer Security Problem: An Overview," *IEEE Computer,* Vol. 10, No. 6, June 1977, pp. 50-62, 71-73.
2. Private communication. See also other references below under Bisbey, Carlstedt, and Hollingworth.
3. McPhee, W. S., "Operating System Integrity in OS/VS2," *IBM Systems Journal,* Vol. 13, No. 3, 1974, pp. 230-252.
4. Robinson, L., K. N. Levitt, P. G. Neumann, and A. K. Saxena, "A Formal Methodology for the Design of Operating System Software," in R. T. Yeh (ed.), *Current Trends in Programming Methodology, Vol. 1: Software Specification and Design,* Prentice-Hall, Englewood Cliffs, NJ, 1977, pp. 61-110.
5. Parnas, D. L., "A Technique for Software Module Specification with Examples," *CACM,* Vol. 15, No. 5, May 1972, pp. 330-336.
6. Löhr, K.-P., "Beyond Concurrent Pascal," *Proc. Sixth Symposium on Operating Systems Principles, ACM SIGOPS Operating System Review,* Vol. 11, No. 5, November 1977, pp. 173-180.
7. Good, D. I., "Constructing Verified and Reliable Communications Processing Systems," *ACM SIGSOFT Software Engineering Notes,* Vol. 2, No. 5, October 1977, pp. 8-15. Excerpted from D. I. Good, ed., Final Report of the Certifiable Minicomputer Project, University of Texas, 1977.
8. Neumann, P. G., R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, *A Provably Secure Operating System: The System, Its Applications, and Proofs,* Final Report, Project 4332, SRI International, Menlo Park CA, 94025, 11 February 1977.
9. Glaseman, S., R. Turn, and R. S. Gaines, "Problem Areas in Computer Security Assessment," *Proc. National Computer Conference,* 1977, pp. 105-112.
10. Biba, K. J., *Integrity Considerations for Secure Computer Systems,* MTR-3153, MITRE Corp., Bedford, MA, June 1975.
11. Lampson, B. W., "A Note on the Confinement Problem," *CACM,* Vol. 16, October 1973, pp. 613-14.
12. Lipner, S. B., "A Comment on the Confinement Problem," *Proc. Fifth Symposium on Operating Systems Principles, ACM SIGOPS Review,* Vol. 9, No. 5, 19-21 November 1975, pp. 192-196.
13. Bell, D. E. and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model,* Vols. I, II, III, MITRE Corp., Bedford MA, November 1973-June 1974.
14. Feiertag, R. J., K. N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," *Proc. Sixth Symposium on Operating Systems Principles, ACM SIGOPS Operating System Review,* Vol. 11, No. 5, 16-18 November 1977, pp. 57-65.
15. Nielsen, N. R., et al., *Computer System Integrity Safeguards: System Integrity Maintenance,* SRI International, Menlo Park CA 94025, October 1976. See also N. R. Nielsen and B. Ruder, "Computer System Integrity Safeguards," *IFIP 77,* North-Holland Publ., 1977, pp. 337-342.
16. Bisbey, R., II, G. Popek, J. Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time,* ISI/SR-75-4, December 1975.
17. Bisbey, R., II, J. Carlstedt, and D. Chase, *Data Dependency Analysis,* ISI/RR-76-45, February 1976.
18. Carlstedt, J., R. Bisbey II, and G. Popek, *Pattern-Directed Protection Evaluation,* ISI/SR-75-31, June 1975.
19. Carlstedt, J., *Protection Errors in Operating Systems: Validation of Critical Conditions,* ISI/SR-76-5, May 1976.
20. Hollingworth, D. and R. Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals,* June 1976.
21. Robinson, L. and K. N. Levitt, *Proof Techniques for Hierarchically Structured Programs,* SRI Report, January 1975, *CACM,* Vol. 20, No. 4, April 1977. Also in R. T. Yeh (ed.), *Current Trends in Programming Methodology: Vol. 2, Program Validation,* Prentice Hall, Englewood Cliffs NJ, 1977, pp. 173-196.
22. Roubine, O. and L. Robinson, *SPECIAL (SPECIfication and Assertion Language): Reference Manual,* SRI Technical Report CSG-45, Third Edition, SRI International, Menlo Park CA 94025, January 1977.
23. Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchell, and G. L. Popek, "Report on the Language Euclid," *ACM SIGPLAN Notices,* Vol. 12, No. 2, February 1977, pp. iii+1-79.
24. Wirth, N., "Modula: A Language for Modular Multiprogramming," *Software Practice and Experience,* Vol. 7, 1977, pp. 3-35.
25. Schroeder, M. D., D. D. Clark, and J. H. Saltzer, "The Multics Kernel Design Project," *Proc. Sixth Symposium on Operating Systems Principles, ACM SIGOPS Operating System Review,* Vol. 11, No. 5, 16-18 November 1977, pp. 43-56.
26. Thompson, K. and D. M. Ritchie, *UNIX Programmer's Manual,* Bell Telephone Laboratories, Inc., Sixth Edition, May 1975.
27. Kampe, M., C. Kline, G. Popek, and E. Walton, *The UCLA Data Secure UNIX Operating System,* Technical Report, UCLA Dept. of Computer Science, Los Angeles CA, July 1977.
28. Neumann, P. G., "Toward a Methodology for Designing Large Systems and Verifying Their Properties," 4. Jahrestagung, Gesellschaft fur Informatik, Berlin, October 9-12, 1974, in *Lecture Notes in Computer Science,* Vol. 26, Springer Verlag, Berlin, 1974, pp. 52-67.

History of programming languages

by JEAN E. SAMMET
IBM, Federal Systems Division
Cambridge, Massachusetts

This panel session will provide information on the planning, problems and results of a conference sponsored by ACM SIGPLAN held June 1-3, 1978, and entitled "History of Programming Languages". This conference is intended (1) to *initiate* the preservation of a historical record for some major current languages and to give impetus to others to continue adding to this record, and (2) to provide information from one or two key contributors to the early technical development of the selected languages.

Thirteen languages were selected for presentation and discussion at this conference, based on the characteristics that the languages (1) were created and in use by 1967, (2) are in use in 1977, and (3) have had considerable influence on the field of computing. The actual criteria for choosing the languages were: usage, influence on language design, overall impact on the environment, novelty, and uniqueness. This is not a conference on the entire history of programming languages, nor even a conference on the entire history of the selected languages; it covers primarily their early developments, with emphasis on the technical aspects of the language design. The languages selected are: ALGOL 60, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, SNOBOL.

The organization of this conference was unusual in several ways. The invited speakers were assisted in preparing high quality papers by giving them about 13 pages of questions which applied to all languages, and which it was hoped would be answered in many of the papers. In addition to that, specific questions pertaining to individual languages were sent to each speaker. Examples of the general questions are "What languages were known to you when you started work on your language and what influence did they have?" and "What class of users was your language designed to help?" and "To what extent did concerns about compilation efficiency affect the design?" Examples of specific questions are "In COBOL why is there a COMPUTE verb allowing formulas, as well as ADD, SUBTRACT, MULTIPLY and DIVIDE verbs to do arithmetic" and "Why wasn't there any input/output included in ALGOL 60?"

The problems of having contemporary technical history written by people who were personally involved but are not trained historians are formidable. They include the difficulty of resisting the temptation to use *only* memory rather than written documents for some material, of avoiding insults to people still living and actively working, of making sure that proper credit is given to co-workers and in general the problem of handling delicate issues involving interpersonal conflicts.

The preprints of the papers are being produced as the August 1978 issue of ACM *SIGPLAN Notices*. Following the conference, a book is planned which will include these papers (which may be revised in the light of experience and comments on the presentation) and the commentary from the audience since the entire conference will be audiotaped for future listening by anyone.

COBOL—The 1980 standard, a preview

by GEORGE N. BAIRD, MARGARET M. COOK and ROGER J. GORG
Department of the Navy
Washington, DC

INTRODUCTION

It is quite reasonable to state that COBOL is the most widely used programming language in the world. Originally defined by the Conference on Data Systems Languages (CODASYL) in 1959, COBOL has received wide acceptance. One of the reasons for this is the standardization of the language which gives the user the anticipated ability of being able to transport COBOL programs from one computing system to another without a massive conversion effort. Other reasons for its acceptance are its easy to understand and English-like structure, its ease in being maintained, and its precise data definition which includes editing capabilities.

The standardization of COBOL was first accomplished in 1968[1] after a six year effort. The subsequent conversion from nonstandard COBOL to COBOL 68 resulted in language conversion programs being produced which aided users in converting to standard COBOL compilers that were not necessarily compatible with their predecessors. The advantage to the user in performing this conversion was to obtain COBOL application programs based upon a stable language specification which offered some guarantee that compilers (vendor's version of the COBOL language), would also have stability between releases.

COBOL 68 was revised in 1974.[2,3] This resulted in areas of incompatibility between the two standards. Some of the incompatibilities could not be avoided. They were the result of ill-defined language specifications having been included in COBOL 68 which were subsequently explicitly defined in COBOL 74 (e.g., the COBOL 68 Random Access module was replaced in COBOL 74 with the Relative I-O and Indexed I-O modules). The reason for other incompatible differences could not be as easily explained. The EXAMINE statement being replaced by the INSPECT statement is an example. The EXAMINE statement could have been enhanced to include all of the capabilities of the new INSPECT statement resulting in compatible programs between the two standards.

The technical committee responsible for the standardization of COBOL, American National Standards Committee (ANSC) X3J4, is in the process of preparing for the revision to COBOL 74. This paper presents a preview of what the revised COBOL standard might be. The earliest date that a new COBOL Standard could be developed and approved is the year 1980, so this paper will refer to the revision of X3.23-1974 as COBOL 80.

THE COBOL STANDARDIZATION PROCESS

COBOL is unique in that one organization, CODASYL, develops the COBOL language and another organization, the American National Standards Institute (ANSI), standardizes the language. This arrangement is beneficial in that two separate groups are involved with COBOL resulting in more immediate exposure than is afforded when a single committee is responsible for both the development and the standardization of a programming language. The negative aspect of the COBOL standardization process is that the language specifications for the revision to the COBOL standard must be already included in the existing standard or must be contained in the CODASYL COBOL Journal of Development (JOD)[4] as of a given date. Thus any new feature must first be approved by CODASYL before ANSI can include it as part of a COBOL standard.

The ANSI technical committee responsible for the standardization of COBOL, X3J4, is currently making preparations to produce the revision to COBOL 74. There are two opinions concerning this revision. One opinion is that the new COBOL standard should reflect the latest CODASYL specifications and that compatibility with the previous standard is not a primary consideration. The other viewpoint is that compatibility between versions of the COBOL standard is extremely important because of the investment in existing programs. There will be cases where compatibility may be sacrificed in order to attempt to keep up with all of the changes made in the CODASYL COBOL Journal of Development since COBOL 74, the previous standard, was produced.

When ANSC X3J4 is considering making changes to the COBOL standard during a revision process there is a specific set of selection criteria which is used in determining the acceptability of each change. The set of criteria is as follows:

(1) The general usefulness of an element or function in terms of:
   a. The degree of implementation;

b. Acceptance by users;
c. The degree to which a function was required.

(2) The overall functional capability of the language, considering such things as redundancy.
(3) The state-of-the-art technology with regard to implementing the language feature.
(4) The usefulness, in terms of application requirements, of language capabilities within each level of a module.
(5) Compatibility with other standards.
(6) The cost of implementing versus advantages of use.
(7) Overall language consistency within a defined level or module.
(8) Upward compatibility of levels within a module.

One must carefully examine each of these eight criteria and ask how the impact of any proposed change to the language is measurable in terms of quantity (dollars and cents), or whether it is measurable only in a qualitative sense (relatively good or bad). Vendors have a large investment in existing software products which include certain language elements and features that are extensions to the current standard. Users, however, have a larger investment in the programs they have produced using those compilers. The emphasis of the revision of a language standard should be for compatibility with the previous standard where possible.

AN ANALYSIS OF CODASYL COBOL SINCE 1972

The cutoff date of the CODASYL COBOL JOD for COBOL 74 was January 1, 1972. That was the latest dated document from which language elements were candidates for inclusion in COBOL 74. The cutoff date for COBOL 80 will be January 1, 1978. Roughly six years of development of COBOL as defined by CODASYL will have taken place between COBOL 74 and COBOL 80.

Since the input to the revised ANSI COBOL standard is limited to what is in the CODASYL COBOL JOD, the majority of the potential changes are already known and can be analyzed in terms of their impact if they are included in the revised COBOL standard. From a negative standpoint, language elements deleted from the COBOL JOD since the COBOL 74 cutoff date could influence the deletion of those elements from COBOL 80 as well. Therefore, within the scope of this paper, we can discuss language elements and capabilities which are either new, changed or deleted. The latter two categories will require program modification in order to produce programs conforming to COBOL 80 from programs which currently conform to COBOL 74.

New language elements which give additional capability

Many of the changes made to the CODASYL COBOL JOD have resulted in additional language capabilities. Some of these changes have relaxed punctuation rules while others have changed the category of some required reserved words to optional reserved words. Other changes are merely cosmetic in nature which could make program writing easier. These general changes will be discussed first.

General language changes

Source program punctuation has been clarified and simplified. The use of the semicolon, comma, and space can be used interchangeably in a source program. In COBOL 68 the semicolon and comma could be used interchangeably but only in certain places. This relaxation of the punctuation rules will permit the semicolon and comma to appear after any word or character-string contained in the source program. This should reduce syntax errors due to the misuse of the punctuation characters semicolon and comma.

The Environment Division which was a required division in a COBOL 74 program is now optional. The Configuration Section within the Environment Division is also optional when the Environment Division is present. This change was made primarily because of the new interprogram communication facility which is described later. However it will simplify the production of source programs by reducing the number of required lines of code which must be produced. The presence of an Environment Division in a called program which has no files in it is a waste of time and effort. If the Configuration Section is specified and the Source Computer and Object Computer paragraph headers are present, but the computer name has not been specified, then the compiler will insert the computer name in the appropriate place(s).

Another change which will reduce unintentional syntax errors in source programs permits user-defined words to be the same as system names supplied by the compiler writer. COBOL 74 prohibited these names from being identical. For reasons of portability and conversion this will reduce problems by requiring the compiler to accept user-defined names which are the same as system names. The compiler in this case uses the context in which the word is found to distinguish between a system name and a user-defined word.

The order of clauses within entries in the Environment and Data Divisions is very rigid in some places while other cases permit the clauses to appear in any order. The adopted changes to the SELECT statement and File Description entries will allow their subordinate clauses to be specified in any order. This will make programming easier and it is the clause ordering procedure which has been adopted for other areas in COBOL. The I-O Control paragraph in the Environment Division and the Communication Description entry in the Data Division now permit their subordinate clauses to appear in any order. Another case of inconsistency that has also been resolved regards words which are sometimes optional and other times required. The COBOL word "IS" is optional throughout COBOL 74 except for the Environment Division. This difference has been resolved by making the presence of the word "IS" optional in the Environment Division as well.

The COBOL character set, after having consisted of 51

characters for 18 years, has been expanded to 59 characters. The new characters are the commercial "@," question mark "?," colon ":," apostrophe "'," ampersand "&," percent "%," number sign "#," and exclamation point "!." The characters which can be used in alphanumeric literals include all of the characters in a given computer's character set. There is no guarantee that the characters outside the COBOL character set would be available on different systems. The addition of these eight characters in the COBOL character set means they will be available on all systems supporting COBOL and may preclude some reprogramming when moving programs and data around.

The sequence area (positions 1-6 of the COBOL source statement) is restricted to containing numeric characters in COBOL 74. The adopted changes will allow it to contain a combination of any characters in the computer's character set. There is no requirement that the source statements be in any sequence based on the sequence area and there is no requirement that the sequence area in any particular statement be unique within the source program. COBOL 74 specifies that nonnumeric literals cannot be longer than 120 characters. This restriction has been lifted in CODASYL COBOL with no upper limit. Presumably if the standard reflects this change it will contain a minimum length that must be supported.

An enhancement in the Data Division which will ease the coding of programs is the change which makes the word FILLER optional. COBOL 74 requires that data items which are not to be referenced directly in a COBOL program be present in such a way that they represent the data being ignored. The data area they represent uses the word FILLER in the place of a data name in the data description entry. The presence of the word FILLER in effect does nothing. Under the adopted changes FILLER will be an optional word so that data areas which must be defined, but will not be referenced directly, need not have either a data-name or the word FILLER specified in their data description entry.

A new technique by which data can be referenced in COBOL has been defined. Reference modification permits the programmer to reference a portion of a data item by naming the data item, by specifying a starting position within the data item, and by associating a length with that particular reference. The starting position and length can be specified through variables. By this method a character string of one or more characters can be easily located within a string of data.

COBOL 74 had a restriction that tables could only be three dimensions (i.e., arrays nested three deep). This restriction has been removed and the number of dimensions a table may have has no upper limit. The standard, if this change is adopted, will probably place an upper limit on the number of dimensions that a table may have.

One last inconsistency has been cleared up. This is in the area of character string continuation. All character strings can be continued in COBOL 74 except for the PICTURE character string. This inconsistency must have been brought about by an oversight during the maintenance of the language. Now at long last, a PICTURE character string can be continued.

The majority of the general changes merely makes the rules for the language consistent throughout COBOL. Other changes are for the programmer's convenience. The only changes which will really give additional capability are reference modification and tables with dimensions greater than three.

Nucleus module additions/enhancements

The Nucleus module is the functional processing module of the COBOL standard. A new capability in the JOD is the Boolean or Bit data item. The Bit data item allows the programmer to define a data item in terms of bits. It can represent one or more bits and is used to represent data which can have only two distinct values, 1 and 0.

The ACCEPT statement has been expanded to include the DAY-OF-WEEK as a conceptual data item from which information can be obtained. DAY-OF-WEEK is composed of a single data element that represents the day of the week. The values obtained from DAY-OF-WEEK are one through seven. One represents Monday, two represents Tuesday . . . seven represents Sunday.

The DISPLAY statement has been modified to provide a new capability for interacting with hardware devices for which positioning has meaning. When the new WITH NO ADVANCING phrase is specified, the hardware device will not be reset to the next line or changed in any way following the transmission of the last operand specified in that DISPLAY statement. If the WITH NO ADVANCING phrase is not specified, then after the last operand has been transmitted to the hardware device, the positioning of the device (e.g., cursor) will be reset to the leftmost position of the next line of the device.

The INSPECT statement has been enhanced in two major areas. A new format of the INSPECT statement contains the CONVERTING phrase. The CONVERTING phrase causes the INSPECT statement to work semantically like the TRANSFORM statement which is supported by many COBOL implementations but is not contained in either COBOL 74 or the COBOL JOD. The data item containing the data to be inspected is named. Two additional operands indicate the characters to be converted and what each character should be converted to.

```
INSPECT DATA-ITEM CONVERTING "AX2" TO
                             "M5B"
        DATA-ITEM (before) "1A2BWHYX"
        DATA-ITEM (after)  "1MBBWHY5"
```

This is a simple way to convert a number of characters in a string of data. COBOL 74 requires a series of "old character" BY "new-character" phases to accomplish the same thing.

A second enhancement to the INSPECT statement allows the inspection to be initiated and terminated within the string

of data being inspected. This is accomplished through the use of BEFORE and AFTER phrases.

```
INSPECT DATA-ITEM REPLACING 'X' BY 'Y'
  AFTER INITIAL 'A' BEFORE INITIAL 'Z'
  DATA-ITEM (before) X1A2X3Z4X5
  DATA-ITEM (after)  X1A2Y3Z4X5
```

Replacement begins after the first 'A' is located and terminates when the first 'Z' is encountered.

Note that the operation of the above example is data dependent. If the 'A' and 'Z' are not encountered in the proper sequence, different results would occur.

The MOVE statement has been modified so that numeric edited data can be moved to a numeric data item and the editing characters/symbols will be removed. This will help with data interchange among other systems and other languages. When the CORRESPONDING phrase is present in a MOVE statement, there can be more than one receiving group specified. COBOL 74 only permits one receiving group.

The PERFORM statement has been overhauled in an attempt to satisfy the syntactic and semantic needs of structured programming. These will be discussed under the general type of changes for structured programming. Other changes in the PERFORM statement parallel the changes to the definition of tables. COBOL 74 allows tables to be up to three dimensions. With the newly adopted changes no upper limit will be specified. The PERFORM statement in COBOL 74 can vary up to three indexes/identifiers, corresponding to the number of dimensions permitted with a table. The upper limit of three varying phrases has been removed, permitting an unspecified number of VARYING phrases. Presumably, there will be an upper limit in COBOL 80.

The SET statement, used strictly to modify indexes associated with tables in COBOL 74, has been expanded to include the ability to change the collating sequence used by the Program, the Sort/Merge process, or the Code-set for files during the execution of the program. It also permits the status of external switches to be changed and the value of conditional variables (88 levels) to be altered.

There are some interesting changes which have been made to the Nucleus module which will make programming easier in some respects. However, no change of any real impact has been accomplished.

Input-output module—Additions/enhancements

A technique for establishing a padding character for sequential files is available for inclusion in COBOL 80. COBOL 74 does not address padding characters, the problems of a short block, or a block that is not completely filled with valid data records. By including the PADDING CHARACTER IS clause in the Select statement for a file, the character specified will be used to fill out that portion of the block that exists beyond the last physical record. Another change is the RECORD DELIMITER IS phrase which can be used to specify an implementor defined technique for producing/identifying variable length records. In COBOL 74 determination of how to produce variable length records and subsequently recognizing them is left to an undefined implementor technique.

The ability to specify an optional file (one which may or may not be present at execution time) is permitted only for sequentially organized files in COBOL 74. This facility has been expanded to include files whose organization is either Relative or Indexed. OPEN. . .EXTEND will also be permitted for Relative and Indexed files. COBOL 74 permits the EXTEND option (which allows records to be added beyond the end of a file) to be specified only for sequentially organized files. Another enhancement to the OPEN statement affects the use of the I-O phrase. COBOL 74 permits the use of the I-O phrase only if a file already exists. This causes some confusion to programmers and it makes little or no sense as a restriction. The file could have been opened for output, one record written, the file closed, and then opened for I-O. Now the use of the I-O phrase can be used for a file that has not been previously created.

The RERUN statement has been modified. This statement is used to trigger a checkpoint dump to be taken during the execution of a program. The file to be used can be either an implementor named file or a file described in the COBOL program. COBOL 74 requires that if the file being used is described by the program that it be a sequentially organized file. The rules also have been changed so that a relative or indexed file can be used as well. The wisdom of using a relative or indexed file for storing checkpoints may be questionable but the capability now exists.

The MULTIPLE FILE TAPE clause found in the Input-Output Control paragraph has been clarified. In COBOL 74 there is a question regarding whether a File Description entry must be present in the program for files which are on the tape but are not going to be referenced. Under the adopted changes the introduction of a pseudo-file-name will be used to identify these files. The new facility also allows a multiple file tape to be contained on more than one reel or physical volume.

Variable length records will be requested using the new form of the RECORD clause in the File Description Entry for those files.

```
RECORD IS VARYING IN SIZE FROM SHORT-
  LENGTH TO LONG-LENGTH CHARACTERS
  DEPENDING ON LENGTH-INDICATOR
```

The minimum and maximum record lengths are specified in the RECORD VARYING clause. The DEPENDING ON phrase names the data item which will be used to define the length of the current record. This data item will be defined in the Working Storage or Linkage Section. Prior to the record being written, the length of the current record is placed in the Depending On data item by the programmer. This establishes the logical length of the record to be written. Later when the record is read, the input/output control system places the length of the record in the Depending On data item. Using this value the programmer then will know the size of the record just read.

The LABEL RECORDS clause has been made optional, whereas COBOL 74 requires the clause be present for each file described in the source program. When the LABEL RECORDS clause is not specified, the file will be treated as though the STANDARD option had been specified. Another change makes the words RECORD/RECORDS optional words in both the LABEL RECORDS and DATA RECORDS clauses.

The REEL/UNIT phrase can be used in a CLOSE statement for a sequentially organized file which is a report file (i.e., produced by the COBOL Report Writer). This is explicitly prohibited in COBOL 74. The usefulness of this function may be questioned since a report file is subsequently passed to the operating system for printing.

The DELETE statement has been modified so that a file can be referenced as well as a record within a file. When the FILE format of the DELETE statement is used, the referenced file is deleted and any associated storage medium is released and made available to the operating system. The file being deleted must not be a sequentially organized file. After the DELETE statement has been executed, the file is no longer available to the Input-Output Control System.

Somewhat related to the Input-Output modules is the Sort-Merge module. Only a few additions have been made. Both the SORT and MERGE statements can have multiple files named in the GIVING phrase. This permits two or more copies to be created of the file produced as a result of a SORT or MERGE operation. In addition, the GIVING phrase of a SORT or MERGE statement now may reference relative or indexed files. COBOL 74 requires that the giving file be a sequentially organized file.

The SORT statement has a new phrase, the WITH DUPLICATES IN SEQUENCE phrase, which will cause records whose keys are identical to remain in the sequence in which they are input to the sort process. The absence of this phrase, as in COBOL 74, leaves the task of sorting records with duplicate key-values to the implementor.

Inter-program communication module additions/ enhancements

One of the two major enhancements to CODASYL COBOL since the COBOL 74 cut-off date is the modification made to the interprogram communication facility (CALL statement). The impetus to make this modification was the recent emphasis on software engineering and structured programming. The other major enhancement to COBOL is the inclusion of the syntax and semantic forms necessary to be able to do structured programming in COBOL directly without having to simulate the major structured programming constructs. The structured programming aspect of COBOL will be addressed later.

COBOL 74 suffers in software engineering circles due to the structure of the language. Data within a given program is available throughout the program. There is no method by which data for a given set of procedures can be earmarked as being available only to those procedures. The JOD COBOL has been modified to provide a solution to these problems.

Source programs can be contained within other source programs. The contained source programs are referenced by the CALL statement. A program will always be in its initial state when called if the program is identified as being an INITIAL program. If a contained program is to be accessible to programs other than the program which contains it, its PROGRAM-ID statement must indicate that the program is COMMON. The END PROGRAM statement terminates a contained program.

```
IDENTIFICATION DIVISION
PROGRAM-ID.  PROG1.
  .
  .
  .
IDENTIFICATION DIVISION
PROGRAM-ID. PROG2 IS COMMON INITIAL
  PROGRAM
  .
  .
  .
END PROGRAM PROG2.
  .
  .
END PROGRAM PROG1.
```

PROG2 is a source program contained in PROG1. When PROG1 calls PROG2 parameters can be passed via the USING phrase of the CALL statement, as was the case in COBOL 74. Further, data in the contained program (PROG2) described as EXTERNAL can be referenced by the containing program and by any other program in the run unit. The data items in the referencing programs must be described in the same manner and using the same datanames and equivalent attributes as were used to define the data items in the called program. Names in the containing and contained programs that are identical, but not identified as being EXTERNAL in the contained program, reference unique items within each program. Data items in the containing program (PROG1) identified as GLOBAL can be referenced by the contained program (PROG2). The names of GLOBAL data items must not be duplicated in other programs referencing them.

There is also an additional option in the passing of parameters when using the CALL statement. If the BY CONTENT phrase is used in naming a parameter to be passed, then the called program can modify its version of the parameter but it has no effect on the value of the parameter in the calling program. If the BY REFERENCE phrase is used in naming parameters, the result is the same as it is in COBOL 74. Any changes to the data in the called program will affect the data in the calling program.

Structured programming enhancement/additions

COBOL 74 has, at best, co-existed with structured programming. The basic control structures are not available in COBOL 74 and have to be simulated using variations of the

PERFORM statement. Additionally, the number of places that asynchronous interrupts can occur in COBOL is well beyond what is suggested in structured programming.

IF-THEN-ELSE and Asynchronous interrupts

The problems associated with the COBOL 74 IF statement have been solved in the JOD COBOL. Pure nesting of the COBOL 74 IF statement was precluded because of the syntax of the language. The END-IF phrase has been introduced to make the IF statement more functional.

```
IF
  Condition
THEN
  statement-1
ELSE
  statement-2
END-IF
```

In COBOL 74 an IF statement is terminated by an ELSE phrase associated with an IF statement at a higher level or by the separator period. Using the JOD COBOL the END-IF phrase can be used to specifically terminate a nested IF statement.

The statements containing asynchronous interrupts have been handled in a similar fashion.

```
ADD identifier TO identifier
  ON SIZE ERROR imperative-1
  NOT ON SIZE ERROR imperative-2
END-ADD
```

Using the ADD statement as an example the ON SIZE ERROR phrase illustrates the true line path of the condition, whereas, the NOT ON SIZE ERROR phrase is the ELSE (or false) side of the condition. The END-ADD phrase terminates the ADD statement and its inter-related conditional statement. The END-verb phrase has also been included in all statements which can cause an asynchronous interrupt (i.e., statements containing the ON SIZE ERROR, AT END, ON EXCEPTION, INVALID KEY, ON OVERFLOW, WHEN, and NO DATA phrases.).

DO-WHILE and DO-UNTIL

The PERFORM statement in JOD COBOL has been expanded to include an inline looping capability.

```
PERFORM WITH TEST [BEFORE]
                  [AFTER ] ...
  .
  .
  .
Procedural Code
  .
  .
  .
END-PERFORM
```

By specifying the TEST BEFORE phrase, the procedural code will be executed as a DO-WHILE construct. If the TEST AFTER phrase is specified, then the procedural code will be executed as a DO-UNTIL construct.

The EVALUATE statment, which is much too complex for the scope of this paper, represents CODASYL's answer to the CASE construct in structured programming. The EVALUATE statement can also be used to express a decision table in a COBOL format.

Incompatible changes to/deletion of language elements

As the CODASYL COBOL specifications evolve changes are made to the existing syntax and semantic actions of language elements; and language elements are deleted. These changes, if reflected in COBOL 80, will require source programs to be modified in some cases quite extensively. The most drastic of the incompatible changes are itemized in the following paragraphs. Language elements which have been deleted from CODASYL COBOL are:

- The 77 level data item. The rationale for this deletion is that an elementary 01 level data item accomplishes the same thing.
- The ALTER statement. The value of using the ALTER statement has been questioned for some time due to the way in which it can obscure the logic of a program.
- Numeric Paragraph and Section names in the Procedure Division.
- The REVERSED phrase of the OPEN statement.
- Initial segments (Segmentation Module) 50-99.

The following language elements have been modified such that their inclusion in COBOL 80 will require source programs to be changed in order to continue to use the function.

- The BLOCK CONTAINS clause, and the CODE-SET clause have been removed from the File Description entry in the Data Division and placed in the SELECT statement of the Environment Division.
- The ACCESS MODE, RECORD KEY, ALTERNATE RECORD KEY and FILE STATUS clauses have been deleted from the SELECT statements in the Environment Division and placed in the File Description entry of the Data Division.
- The rules for the RECORD CONTAINS clause state that the absence of the clause will cause fixed length records to be written regardless of the record descriptions. COBOL 74 treated the RECORD CONTAINS clause as a comment and used the actual record descriptions to determine the length of records.
- SORT or MERGE files cannot be named in a RERUN statement to trigger a check point to be taken during the execution of a Sort or Merge operation. Also, a SORT or MERGE statement cannot be referenced in a USE FOR EXCEPTION statement in the DECLARATIVE portion of the program.
- For the INSPECT statement (remember the EXAMINE

statement), both TALLYING and REPLACING cannot be included in the same statement as was permitted in COBOL 74.

- The description of the DEBUG-ITEM associated with Debugging procedures has been changed.
- All files included on a multiple file tape must have the same labeling scheme, i.e., STANDARD or OMITTED. COBOL 74 has no such restrictions.
- The REDEFINES clause cannot be specified in the Data Description entry for an index data item.

The changes described in this section, with the exception of the deletion of the ALTER statement, are questionable. If all of these changes are reflected in COBOL 80, every COBOL program ever written would have to be modified to be acceptable to a COBOL 80 compiler. There is a possibility that X3J4 will allow a transition period for some of the incompatible changes; for example, the clauses that have been moved from the File Description entry to the SELECT statement in CODASYL COBOL will appear in both places in COBOL 80. Language elements in COBOL 80 will be annotated to indicate that they will be deleted when COBOL 80 is revised. This will give the COBOL user advanced warning about potential changes to COBOL which impact existing programs.

CONCLUSION

The revision cycle for developing COBOL 80 from COBOL 74 and the 1978 JOD is under way. It seems appropriate at this point to ask two basic questions. Based on the enhancements which can be included in COBOL 80, is the industry ready for a new COBOL standard? What would COBOL users gain by adoption of a new standard?

COBOL compilers based on COBOL 74 have only been available since late 1975 and early 1976, and some vendors had not released COBOL 74 compilers by the end of 1977. Most of the COBOL community has not yet undergone the conversion effort required for changing COBOL 68 programs to COBOL 74. After they have gone through this process, most will not be ready to accept another conversion effort five years later.

Once a revision cycle is started, it develops its own momentum, and the question of whether the user community is ready for a new standard may be lost. With the exception of reference modification, variable length records, and structured programming changes, there does not appear to be any enhancements justifying the program modifications which would be required, and the increase in size and cost of COBOL compilers. The COBOL community has the responsibility to communicate their views on these questions to the American National Standards Institute.

REFERENCES

1. *American National Standard COBOL, X3.23-1968,* American National Standards Institute Incorporated, New York 1968.
2. *American National Standard Programming Language COBOL, X3.23-1974,* American National Standards Institute Incorporated, New York 1974.
3. Cook, M. M. and G. N. Baird, et al., "An Overview of the 1974 COBOL Standard," *Proc 1975 NCC,* AFIPS Press, Volume 44, pp. 301-307.
4. *CODASYL COBOL Journal of Development 1976* (page changes through May 1977), Department of Supply and Services, Ottawa 1976.

Data Base Facility for COBOL 80

by MARGARET M. COOK
Department of the Navy
Washington, DC

INTRODUCTION

The American National Standards Committee (ANSC) X3J4 is responsible for developing the standard for the programming language COBOL. ANSC X3J4 is currently in the process of revising American National Standard Programming Language COBOL, X3.23-1974.[1] As part of this revision process, X3J4 is developing a candidate Database module for inclusion in the next COBOL Standard. The earliest date that a new COBOL Standard could be developed and approved is the year 1980, so this paper will refer to the revision of X3.23-1974 as COBOL 80.

Direction for developing the COBOL Standard is given to X3J4 by its parent committee, ANSC X3, the ANSI technical committee responsible for Computers and Information Processing. According to X3 direction, the source of the language specifications for COBOL 80 must be either the current COBOL Standard or the CODASYL COBOL Journal of Development.[2] Thus the only possible candidate for standardization of a Data Manipulation Language (DML) for COBOL is the CODASYL database facility. X3J4 cannot consider any other database model, such as relational, for standardization.

In order to develop a candidate Database module for COBOL 80, X3J4 has established a Database Task Group, X3J42, which has the responsibility of reviewing the database facility specifications in the CODASYL COBOL Journal of Development (JOD) and identifying those elements which are to be included in the candidate Database module. The JOD cutoff date for COBOL 80 has been established as January 1, 1978. Database elements in the JOD as of the cutoff date are the source of the database specifications which can be included in COBOL 80.

This paper discusses the structures which can be defined according to the CODASYL database specifications and the language elements which are available for standardization. The standardization effort by X3J4, and hence its subcommittee X3J42, is limited to that portion of a Database Management System (DBMS) which pertains to the defining of a Sub-schema and the Data Manipulation Language statements proposed for the host language COBOL. The portions of a DBMS which are outside the purview of COBOL, such as the Data Definition Language and the Device Media Control Language, are not candidates for standardization by X3J4. However, the functions and structures of the Data Definition Language, because of their close relationship to the Sub-schema definition, are discussed in this paper.

OVERVIEW OF THE DATA BASE FACILITY

The portions of the CODASYL database facility which are of concern to X3J4 are the database functions which permit an application program written in COBOL to communicate with the Database Management System. The functions required are a definition of that portion of a data base of interest to an application program and the procedural statements required to access the database.

The entire database is defined in the Schema. The Schema contains a description of inter-record relationships and the data items comprising each record in the database. Because the Sub-schema definition is a subset of the Schema definition, understanding the Schema and its capabilities is a prerequisite for understanding the Sub-schema definition.

The COBOL Sub-schema defines the portion of the database available to an application program. The part of the database that is not required by a given application area is not defined in the Sub-schema. The Sub-schema definition is a subset of the Schema definition but can define data items in a different manner than the Schema, as long as the definition is consistent with the Schema.

The COBOL Data Manipulation Language statements for the database facility contain update functions for data items and record relationships, and record and item retrieval functions. The DML statements are contained in the Procedure Division of a COBOL program and can only reference records and record structures which have been defined in the Sub-schema referenced by the COBOL program.

The Schema, COBOL Sub-schema and Procedure Division DML statements combined provide a database facility through the host language COBOL. The succeeding sections of this paper discuss each of these components in detail.

THE SCHEMA DEFINITION

The logical definition of the database is provided by the Schema. The Schema defines the record types in the database, the inter-record relationships, and the access control

mechanisms. The specifications for the Schema definition are contained in the CODASYL Data Definition Language Journal of Development.[3]

Record definition

The basic unit of access in the CODASYL database facility is the record. Each record type in the database is defined in the Schema. The record definition specifies the data items and data aggregates which comprise records of a particular record type. The smallest unit of data is the data item. In data subentries within a record definition, complete descriptions are given for each data item in the record. Among the data characteristics specified are the data type, that is, arithmetic, character or bit string, system generated identifier (database key), or implementor-defined extension; validation values for checking accuracy of data item contents when storing items in the database, and access control locks for retrieving, storing, or modifying data items. A named collection of data items in a record type is called a data aggregate and permits the definition of vectors and repeating groups.

In addition to data items whose contents are supplied by the user, the definition of derived data items is also permitted in the CODASYL Schema. Derived data items are data items whose contents are generated by the DBMS according to specifications in the data subentry. There are two types of derived data items, actual and virtual. The DBMS generates the contents of an actual derived data item during a store or modify and the contents are stored as part of the record. A virtual derived data item is generated by the DBMS when a record is retrieved from the database. The item does not exist in the database but is available to a user program as if it were stored in the database.

Within the Schema definition an arbitrary number of record types may be defined, and there is a record description for each type of record in the database. For any given record type defined in the Schema, there may be none, one, or an arbitrary number of occurrences of records in the database.

Set definition

A Set is the basic structure by which inter-record relationships are defined in a CODASYL Schema. Unfortunately, the use of the word set is confusing to those familiar with set as a mathematical entity. The misuse of the letters "SET" according to Steele is "That linguistic atrocity perpetrated by the DBTG Report wherein the nineteenth, fifth, and twentieth letters of the Roman alphabet are used in that order as the name of a peculiar object. This may seem harsh, but the point at issue represents a prize example of the manner in which the information processing sciences generate confusion for themselves and others by casual misuse of words."[4]

A set type in the Schema is defined by a single owner record type and one or more member record types. A record type may be defined to be the member of more than one set type, and a member of one set type may be the owner of a different set type. The set type relationship permits the definition of network structures in the CODASYL database facility.

There are restrictions on the record relationships permitted in the Schema set definition. A record type cannot be defined as both owner and member of the same set type, and a set type may not have more than one owner record type. Thus the many-to-many owner-member relationship is not directly supported by the CODASYL structure.

Set membership

There are two classes of set membership defined for each record type. The first class concerns the insertion of records into a set, and the second class concerns the removal of records from a set. For the insertion of records the member record types can be either automatic or manual. If automatic membership is specified, membership in a set is established by the DBMS when a record is created in the database. If manual membership is specified, a Data Manipulation Language statement is required to connect an already existing record in the database to its owner record.

For the removal of records from a set, the member record types can be either mandatory or optional. If mandatory membership is specified, once a record is a member of a set it cannot be removed from membership in that set type. If optional membership is specified, a member record can be disconnected from a set of the given set type by the execution of a DML statement.

Set selection

The method for the selection of a specific set from the totality of sets in the database is specified in the set selection criteria. The identifiers are specified which are to be used in navigating from owner record to member record in a series of sets on a continuous path, where the member of a given set is the owner of the next set in the path. The values of the named data items are used to select a specific set in the database.

Set order

Each set type defined in the Schema must have an order specified for the member records of a set of the given set type. The member records may be ordered by ascending or descending keys. The keys can be data items, member record names, or system generated identifiers. The member records may also be ordered based upon the insertion order of new member records into the set, for example, first, last, or before another member record selected by an application program.

REALMS

In the Schema the database can be defined to be logically divided into realms (areas). A realm is similar to a COBOL file in that a Data Manipulation Language statement is required before a program can access records within a given realm. The record type definition specifies the realms in which the records of a particular type may appear. An individual record can be associated with only one realm, and a record may not change realms.

There may be an arbitrary number of realms defined in the Schema, and records can be assigned to realms independently of their inter-record relationships (set structure). An owner record and member records of a set may be contained in the same realm or in many different realms. Similarly, a given record type or set type may have occurrences which appear in the same realm or in many different realms.

COBOL SUB-SCHEMA DESCRIPTION

A COBOL Sub-schema definition specifies the data items, record types, set types and realms which can be accessed by a COBOL program. A COBOL Sub-schema contains an individual program's view of the database. Only those data items, records, sets, and realms described in a COBOL Sub-schema may be accessed in a COBOL program referencing that Sub-schema. The portions of a database beyond the scope of an individual program are not defined in the Sub-schema referenced by the COBOL program and are not accessible to the program.

There may be an arbitrary number of COBOL Sub-schemas described for the Schema, and definitions for different Sub-schemas may reference the same Schema entries. A COBOL program references only one Sub-schema, but an arbitrary number of programs may reference the same Sub-schema.

COBOL Sub-schema relationship to schema

A COBOL Sub-schema definition is a subset of the database description given by the Schema. A Sub-schema specifies the data items, records, sets and realms which can be referenced by a COBOL program. A Sub-schema cannot define new record types or set types, and all data defined in the Sub-schema must have been previously described in the Schema.

Important differences in the definitions for the Schema and COBOL Sub-schema are permitted in the CODASYL database facility. The Database Management System uses the Schema and COBOL Sub-schema definitions to perform the data conversion required for access by a COBOL program.

The Sub-schema references data items and records in the Schema by their names given in the Schema. The names for identical data items in the Sub-schema and Schema can differ if they are the subject of an Alias Description entry, in which case use of the Sub-schema names is equivalent to use of the Schema names. The COBOL program always references the data-names defined in the Sub-schema.

Only a portion of a Schema record need be defined in the Sub-schema record and the data items in the Sub-schema may be specified in a different order. The data mapping is performed by matching data-names and record-names to the data-names and record-names in the Schema. The mapping is performed at the elementary item level, and the data description in the Sub-schema is the description used by the COBOL program.

The characteristics of data items defined in the Sub-schema can differ from the Schema specifications as long as the results are consistent with the Schema definition. Data transformation of equivalent data items is performed according to the MOVE statement rules for elementary sending and receiving items. Alternately, a database procedure could be defined in the Schema which performs the required transformation, in which case the MOVE statement rules do not apply.

Data item validation is performed by the DBMS as it stores data into the database or retrieves data from the database. The data item validation values specified in the Sub-schema may be different from the validation values specified in the Schema, but the Sub-schema specification must be within the range of the Schema specification. The DBMS checks the contents of data items when retrieving them from the database and does not retrieve items outside of the Sub-schema specified range.

Access control locks for a given realm, set type, record type, or data item can be declared in either the Schema or Sub-schema, or in both. If an access control lock for a given entity has been declared in both the Schema and Sub-schema, the Sub-schema access control lock overrides the Schema lock and the Sub-schema access control lock must be satisfied by the COBOL program.

The set description in the Sub-schema permits the specification of the arguments to be used by the DBMS in defining the set selection criteria. If set selection criteria are defined for a record type in the Sub-schema, they replace the set selection criteria defined in the Schema. If set selection criteria for a record type are not defined in the Sub-schema, then the Schema set selection criteria remain in effect.

Data independence

Although the data items defined in the Sub-schema must be defined in the Schema, important differences are permitted which give COBOL programs data independence. The definition of data in the database is contained in the Schema which is a step removed from the COBOL program's view of the database as given in the Sub-schema. Because of the differences between the Schema and Sub-schema which are permitted in the CODASYL database facility, changes can be made to the Schema without making any modifications to the programs accessing the database.

Modification of the Schema may require modification of the Sub-schemas based on the Schema. If the changes redefine the set structures or remove data items from a record type, then changes to the Sub-schemas referencing these structures and items must be made. If the changes are the addition of new data items, record types, and set structures which are not referenced by the Sub-schema, or if the data descriptions are changed, Sub-schema modifications are not required.

COBOL Sub-schema divisions

A COBOL Sub-schema source program consists of three divisions: Title, Mapping, and Structure. The Title Division specifies the name of the Sub-schema and the name of the Schema to which the Sub-schema is related, the Sub-schema access control key which must satisfy the Schema access control lock in order for the Sub-schema to be placed in the Sub-schema library, and the access control locks associated with the use of the Sub-schema. The Mapping Division contains the Alias Section where Sub-schema names not defined in the Schema are equated to names defined in the Schema. Identifiers, realm-names, record-names, and set-names may be named in the Alias Section.

The Structure Division contains the Realm Section, the Set Section, and the Record Section. This division specifies the portion of the database that is to be made available to a COBOL program and the structure of that portion of the database. The Realm Section specifies the Sub-schema realms and the access control locks associated with the use of the realms. The Set Section specifies the Schema set types to be made available to the Sub-schema, the set selection criteria, and access control locks associated with the use of the Sub-schema sets. The Record Section specifies the Schema record types and the data items within the record types that are to be made available to a Sub-schema. Access control locks associated with the use of records and data items, and the data item characteristics are also specified in the Record Section.

Compilation and binding time of COBOL Sub-schema

The Schema, COBOL Sub-schema and COBOL program are required to operate as a unit in order to perform database functions. The time of compilation of the Sub-schema and the binding time of the Schema, Sub-schema, and COBOL program as a run unit are extremely important. Some of the possible times for binding are COBOL compile time, COBOL Sub-schema compile time, execution time, or any other implementor-defined time.

The results for a program using the database facility could vary depending upon the binding time. For example, if a Sub-schema and COBOL program are compiled and bound together with a Schema as a run unit, and the Schema definition is then changed, the results of the execution could be quite different from compiling and binding the Sub-schema and COBOL program with the Schema after the changes are made to the Schema.

COBOL 80 Sub-schema subsetting

If a COBOL database facility is included in COBOL 80, it is quite probable that the COBOL 80 Sub-schema will be a subset of the COBOL Sub-schema specifications in the CODASYL COBOL Journal of Development. The X3/SPARC Database Systems Study Group has expressed concern over declarations in the Schema being overridden by declarations in the Sub-schema. Votes taken by ANSC X3J4 indicate that this concern is shared by X3J4.[5] It is probable that the specifications for access control locks for realms, set types, record types and data items will not be contained in the COBOL 80 Sub-schema. If access control locks were permitted for these resources, they would take precedence over access control locks specified in the Schema for the same resources.

The set selection criteria specified in the Sub-schema replaces the set selection criteria in the Schema and may not be included in the COBOL 80 Sub-schema for that reason. In the case of the set selection criteria, there is sentiment in X3J42 that the Sub-schema override of the Schema is not a problem and set selection criteria should be included in the COBOL 80 Sub-schema.

DATA DIVISION SUB-SCHEMA SECTION

If the database facility is included in COBOL 80, the Sub-schema Section will be added to the Data Division. The Sub-schema Section consists of a Sub-schema entry which specifies the COBOL Sub-schema that is to be accessed by the COBOL program, and the access control key which must satisfy the access control lock associated with the Sub-schema.

Only one Sub-schema entry is permitted per COBOL program. Multiple Sub-schemas per program are not presently permitted by the CODASYL Journal of Development and hence cannot be permitted by COBOL 80.

PROCEDURE DIVISION DML STATEMENTS

The Procedure Division Data Manipulation Language statements provide an interface to the database for a COBOL program. The Procedure Division DML statements provide the capability to update the database, to retrieve records and data items from the database, and to control the availability of realms to the COBOL program.

Update function statements

There are DML statements which provide update functions for both set structures, records, and data items within a record. Records are stored in the database by executing a STORE statement. The record stored becomes a member

of each set type in which it has been declared in the Schema to be an automatic member, and the record becomes the owner of an empty set (set with no members) for each set type for which it has been defined to be an owner. Records are removed from the database by executing an ERASE statement. The ERASE statement can remove one record, or with the use of additional options, all records within a set structure for which the current record is the owner record.

A record becomes a member in one or more sets for which the record type has been defined to be a manual member by executing a CONNECT statement. A record is removed from one or more sets for which its record type has been defined to be optional by executing a DISCONNECT statement.

The MODIFY statement is used to change the contents of one or more data items in a record, the set membership of a record, or to change both data items within a record and the set membership of a record. Execution of a MODIFY statement without the ONLY option causes the DBMS to replace in the database the contents of data items within the current record with the contents of the record area associated with the record. Derived data items may be indirectly changed by execution of a MODIFY statement. If the INCLUDING option or the ONLY option is specified in a MODIFY statement, the set membership of the current record is changed in accordance with the set ordering criteria of each set type.

The ORDER statement permits the logical reordering of members of the set of which the current record is either an owner or a member record. The logical reordering may apply to only the current run unit, or the reordering of the set member records may occur in the database. The object records of an ORDER statement may be all the records referenced by a given record-name, all records in which a given data-name is defined, or all records that are members of the set containing the current record.

Retrieval function statements

The two DML statements which are used to locate records and retrieve all or part of a located record are the FIND and GET statements. The FIND statement is used to locate a specific record in the database which then becomes the current record of the run unit. Execution of the FIND statement does not retrieve the selected record but identifies it as the current record for use in subsequent DML statements. The record selection expression in the FIND statement specifies the criteria to be used in determining the object record of a FIND statement. The record selection criteria may be based on record contents, system generated identifiers, or a record's relationship with other records.

The GET statement is used to move all or part of a record from the database into its associated record area where it can be accessed by other Procedure Division statements in a COBOL program. The GET statement without any identifier specified or with identifier consisting of a record-name, moves the portion of a database record defined in the Subschema into its associated record area. If identifiers referencing data items are specified, the contents of those data items are moved into the associated record area. Data items not referenced are unchanged in the record area. In all cases the current record of the run unit is the database record accessed.

Control function statements

There are two DML statements which control a COBOL program's access to database realms. The READY statement prepares one or more realms for processing and specifies whether the records in a realm can be accessed and modified, or only accessed. The READY statement also indicates whether the records in a realm are for the exclusive use of the run unit, or if other run units are allowed to access the realm but prevented from updating the specified realm.

The FINISH statement terminates the availability of one or more realms to the run unit. The individual realms to be terminated can be specified by name, or a set name can be referenced in which case all realms which contain the owner or members of the named set are terminated.

The DML includes the COBOL IF statement with database conditions as the conditional expression to be tested. There are three database conditions which may be tested: Tenancy, Member, and Nullity. The Tenancy condition determines if the correct record of the run unit is the owner, member, or either an owner or a member of a set of a given set type. The Member condition determines if a given set has any member records. The Nullity condition is used to determine if a specified data item has the null attribute.

USE declarative statement

The USE declarative statement for the COBOL data base facility specifies the procedures to be used in producing access control keys for satisfying Schema and Sub-schema access control locks pertaining to functions performed on realms, records, sets or identifiers. The USE statement also specifies the procedures to be executed on database exception conditions.

Concurrent run units

There are three DML statements which are included in the COBOL database facility to monitor concurrent usage of records by different run units. The three COBOL statements are KEEP, FREE, AND REMONITOR. The entire area of monitored mode and the functions of these verbs are expected to change from those specified in the Journal of Development in the Fall of 1977, so these statements will not be further explained. ANSC X3J4 has agreed that the KEEP, FREE, and REMONITOR statements, and the extended monitored mode as they are specified in the Fall 1977 COBOL JOD are not suitable for standardization and hence will be excluded from COBOL 80.

COBOL 80 DML statements subsetting

As mentioned earlier in the Sub-schema explanation, ANSC X3J4 is concerned about Sub-schema declarations replacing Schema declarations, and Sub-schema declarations being overridden by programmer action. Votes taken by X3J4 give an indication of the possible subsetting which will occur in determining the DML functions suitable for standardization.[5,6] The ORDER, KEEP, FREE, and REMONITOR statements have been identified as statements which will not be included in COBOL 80. Also, it is probable that the USE statement for data base functions will be subset for COBOL 80.

There are many problem areas which CODASYL is trying to resolve before the cutoff date. Among the data base features which have been identified as problem areas are the NULL concept (Is NULL a value or an attribute?), the results of MODIFY on derived data items not available to the Sub-schema, and the cascading effect of the ERASE ALL statement. Some of these problems may be eliminated by subsetting the database facility.

CONCLUSION

There are many obstacles in the path of the standardization of a database facility for the language COBOL. The Database Task Group X3J42 must develop a candidate Database module based on the CODASYL COBOL Journal of Development 1978. The proposed module must then be approved for publication for public comment as a draft proposed standard, either as a separate document or as a module of COBOL 80. The public comments are reviewed by X3J4, and changes made to the draft Database module based upon the public comments. The final Database module specifications must then be approved by letter ballot by X3J4 and their parent committee X3. After approval by X3, the Database Module is forwarded to the ANSI Board of Standards Review for approval of the publication of the Database module as an American National Standard.

Even if a Database module is approved for inclusion in COBOL 80, much work is still required before there will be a database standard. The inclusion of a database facility in COBOL 80 would only standardize the COBOL Sub-schema and COBOL Data Manipulation Language statements. There would then exist the anomaly of a standard for the COBOL database facility without a standard for the Data Definition Language. Standardization of a COBOL database facility is meaningless without a Data Definition Language standard.

REFERENCES

1. *American National Standard Programming Language COBOL, X3.23-1974,* American National Standards Institute Incorporated, New York 1974.
2. *CODASYL COBOL Journal of Development 1976* (page changes through May 1977), Department of Supply and Services, Ottawa 1976.
3. *CODASYL Data Definition Language Journal of Development 1973,* National Bureau of Standards Handbook 113, U.S. Government Printing Office, Washington, D.C. 1973.
4. Steele, T. B., "Data Base Standardization A Status Report," *IFIP TC-2 Working Conference,* January 1975.
5. Minutes ANSC X3J4 Meeting Number 90, CBEMA, Washington, D.C., July 1976.
6. Minutes ANSC X3J4 Meeting Number 91, CBEMA, Washington, D.C., September 1976.

COBOL—Its relationship with other American national standards

by L. ARNOLD JOHNSON, PATRICK M. HOYT and GEORGE N. BAIRD
Department of the Navy
Washington, DC

INTRODUCTION

The standards which interface with COBOL can effect the degree of machine independence a COBOL program has. COBOL 74[1] provides the capability to process the standard coded character set defined by the American National Standard Code for Information Interchange (ASCII).[2] In that regard, COBOL 74 relies on the ASCII Standard for defining the character codes and the collating sequence that are to be reflected by the program when processing that particular character code. Additionally, COBOL can specify that ASCII be recorded on external media such as punch cards[3] or magnetic tape[4] with magnetic tape labels.[5] If that data is to be used for information interchange, then the recording of the data should be in accordance with the respective media standards. The American National Standards which interface with COBOL must be properly defined and implemented on a computer system for a COBOL program to be independent of the computer system. In the process of validating COBOL compilers, we use these other American National Standards related to COBOL and have found that data cannot be easily transported between computer systems.

ANSC X3 ROLE IN COBOL AND RELATED STANDARDS

American National Standard Committee X3 through its three standing committees is responsible for the standardization of COBOL and other subjects related to systems, computer equipments, devices and media for information processing.[6] Three committees assist X3 in discharging its responsibilities concerning the administration, evaluation, allocation and scheduling of standards projects. The Standards Planning and Requirements Committee (SPARC) and the International Advisory Committee (IAC) of X3 have staff responsibilities while the Standards Steering Committee (SSC) has line responsibilities. The IAC is responsible for coordinating the work of ANSC X3 with respect to international organizations. The purpose of SPARC is to review the need for standards, make recommendations to X3 on the desirability of standards and determine that work of the technical committees is responsive to the original objective of the standardization project. The administrative arm of X3 is the Standards Steering Committee which monitors and coordinates the standards projects that are being developed by the project groups. Individual projects are classified as being in one of three project groups, hardware, software or systems. The technical committees (established for each individual project) designated for the development and maintenance of American National Standards for COBOL, character codes, magnetic tape labels, and data representation are part of the software group. Technical committees for the standardization of physical media associated with magnetic tape and punched cards are included in the hardware group.

The basic objectives of the American National Standards Institute (ANSI) as well as its subordinate committees are the (1) promulgating of standards, (2) coordination of standards development, (3) establishment of standards, and (4) exchange of information. The underlying purpose of the COBOL standard is to "promote a high degree of machine independence in such programs in order to permit their use on a variety of automatic data processing systems."* Further, one of the criteria used by ANSC X3J4 (technical committee for COBOL standardization) for considering a COBOL element as a candidate in the 1974 COBOL standard was its compatibility with other standards. A philosophy similar to COBOL is reflected in the standards associated with magnetic tape, punched cards, magnetic tapes labels and standard coded character set in that these standards are concerned with a common format for information interchange between computer systems. Based on experiences noted later in this paper, there is a question as to whether COBOL and related standards, as yet are fully effective in providing information interchange.

STANDARDS INTERFACING WITH COBOL

There are five different American National Standards which may either directly or indirectly interface with American National Standard Programming Language COBOL,

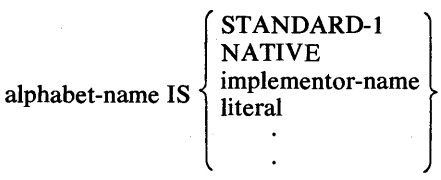
* See Paragraph 1.1, page I-1 in Reference 1.

X3.23-1974. The COBOL 74 Standard conveniently includes the language syntax which make these interfaces possible and thereby provides a common definition for the purpose of data interchange. American National Standard Code for Information Interchange (ASCII) is the common character code used for the interchange of data. Ideally, a standard data interchange on punched cards or magnetic tape is possible between computer systems when ASCII together with the appropriate American National Standards for punched cards or for recorded magnetic tape and magnetic tape labels are used. For data interchange as well as program interchange to be possible, a fundamental assumption is that each of the computer systems between which the data interchange is to occur must implement and support the applicable ADP standards.

Until COBOL 74, there was no definition within the COBOL language which allowed for the designation of a program collating sequence or how data would be represented on external media. Without such a definition each implementor was permitted to use his own techniques regarding the character collating sequence used, recording of signed data on external media, and the character code used for external media. The facilities in COBOL 74 for defining the collating sequence, the character code and representation of signed data are the COBOL features associated with the PROGRAM COLLATING SEQUENCE clause, CODE-SET clause, alphabet-name clause and SIGN SEPARATE clause. For the first time, it is possible for COBOL programs to read, process and produce data that is interchangeable in accordance with related ADP standards.

COBOL 74

The COBOL language feature by which ASCII is written to or read from an external media is the CODE-SET clause plus an associated alphabet-name clause. The CODE-SET clause in an optional clause of the File Description Entry in a COBOL program. When it is present, this clause specifies the character code convention used on the external media. Any conversion that may be required of the code, as represented internally in the computer system, and the code as represented on the external media occurs during execution of either an input or output operation (a READ or WRITE statement). The construct of the CODE-SET clause is "CODE-SET IS alphabet-name." Alphabet-name is defined in the SPECIAL-NAMES paragraph and has the form:



When the STANDARD-1 phrase is specified, the character code or collating sequence identified is that defined by the American National Standard Code for Information Interchange, X3.4-1976. Each character of the standard character set is associated with a corresponding character of the native character code. Note the COBOL Standard does not require that the computer system's architecture use ASCII, only that the computer system and associated COBOL compiler process the character code as if it were ASCII.

The ASCII character code standard defines the bit configuration for representing characters in the standard data format. The value of numeric fields may be represented in either binary or decimal form depending on the equipment. If more than one type of numeric representation is available for a computer system, this selection may be specified through COBOL by use of the USAGE clause. The two options of the USAGE clause are COMPUTATIONAL (binary) and DISPLAY (decimal). Therefore, in order to produce ASCII on an external media via COBOL, data descriptions for the file must be explicitly or implicitly defined as DISPLAY.

Another factor which effects the ability to interchange data between systems is the representation of signed data. The positive or negative characteristic of a numeric value may have, as many possible representations. Signs may be represented as part of the bit configuration associated with the high order or low order position of the numeric value. Signs also may be represented as a separate character which may be either leading or trailing. Since the ASCII character code does not provide for signed data values, the sign characteristic of a numeric value must be carried as a separate character. This is defined through COBOL using the SIGN SEPARATE clause.

Data is only interchangeable if the sequence of data is characteristic of the character code selected and the COBOL program processes it accordingly. This sequence is controlled within the COBOL program by use of the PROGRAM COLLATING SEQUENCE IS alphabet-name clause located in the OBJECT-COMPUTER paragraph. The PROGRAM COLLATING SEQUENCE clause is optional, but if specified, the program uses the collating sequence defined by a corresponding alphabet-name clause. If the alphabet-name clause specifies the ASCII character code (alphabet-name option STANDARD-1) then all nonnumeric comparisons for any relation conditions during program execution should reflect the ASCII code collating sequence.

AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII)

ASCII is the standard coded character set to be used for information interchange among information processing systems. The ASCII Standard describes the code insofar as the bit arrangement or configuration relates to an associated graphic character. A minimum of 7 bits is used to represent an ASCII code character. The Standard does not define the means by which the code set is to be recorded in any physical medium but there is a bit relationship with the ASCII code to other standards such as the American National Standard Recorded Magnetic Tape for Information Interchange, X3.39-1973 (1600 CPI, PE) and also X3.22-1973 (800 CPI, NRZI). For COBOL to produce or recognize ASCII data, the language feature CODE-SET in combina-

tion with the STANDARD-1 option of the alphabet-name clause is required.

The binary bit pattern of the character code gives a collating sequence position for each character in the code. For an implementation to support the ASCII standard, the computer system must be capable of not only accepting and producing ASCII, but also must be able to process ASCII coded data in the proper collating sequence.

AMERICAN NATIONAL STANDARD HOLLERITH PUNCHED CARD CODE X3.26-1970

The Hollerith Punched Card Code Standard specified 256 Hole-pattern in twelve-row punched cards. These Hole-patterns are assigned to the 128 characters of 7-bit ASCII and to 128 additional characters for use in 8-bit coded systems. This standard relates these punched Hole-patterns to a particular bit pattern in an 8-bit data system. An example of such a system noted in this standard is the American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI), X3.22-1973. In addition to relating Hole-patterns to a particular bit construct, the code table presented in this standard also associates a graphic character to that particular position within the table. Columns 0-7 of the table correspond to the bit pattern and associated graphic character of that defined in the ASCII standard. The standard does not specify a card sorting sequence; however, when we consider the bit patterns of any two ASCII characters there is a well-defined sequence relationship.

Since the CODE-SET clause gives COBOL the capability to specify the character code convention used to represent data on the external media (such as punched cards) a relationship does exist through the ASCII character code standard for a COBOL program to accept, process, and create data on punched cards in accordance with X3.26-1970.

AMERICAN NATIONAL STANDARD MAGNETIC TAPE LABELS FOR INFORMATION INTERCHANGE, X3.27-1976

The magnetic tape label standard gives specifications for the recording of magnetically recorded labels on magnetic tapes used for information interchange. Rather than specify how the labels of the tape are to be processed or the data bit structure on the tape, this Standard identifies the information and the structure of tape labels that are recorded on tape. As such, the tape label standard is applicable to either 7 track or 9 track magnetic tape.

The standard makes note of the fact that the physical recording size of labels may be larger than the required 80 characters subject to hardware constraints. These 80-character records are grouped together based on their respective locations on tape related to beginning of tape volume, beginning of file, end of the file and end of tape volume. A tape mark, both before and after the file data, separates a label record group from the file data. Each label record is uniquely recognized by the first four characters in that record. A record group may have one or more records depending on the implementation and an agreement between the data interchange parties. For a single volume tape all label records are optional except for label records identified as VOL1, HDR1 and EOF1 which respectively define the beginning of the volume or tape, header label for beginning of each data file, and label record immediately following the data file. An illustration follows:

VOL1----HDR1---*File Data*[†]
EOF1----**

The tape label standard identifies the information and the data category (numeric or alphanumeric) that is to be stored in each field of the label record. ASCII is mentioned by the standard as a character code which can be used; however, the standard is applicable to any code structure. Certain fields of the label records are designated as optional by the standard. These optional fields may be recognized and processed or may be ignored at the option of the implementor.

COBOL includes provisions for specifying labels for data files through the LABEL RECORD STANDARD clause of the File Description Entry. However, COBOL 74 does not describe the format, structure, or content of file labels. It states only that file labels will be present when the clause is specified. If an implementation claims support of the tape label standard for use with COBOL, then the implementor should be expected to accept and process ANS tape labels on input, process those tape labels and create ANS tape labels on output, both in accordance with the magnetic tape label standard, X3.27-1976.

AMERICAN NATIONAL STANDARD RECORDED MAGNETIC TAPE FOR INFORMATION INTERCHANGE (800 CPI, NRZI), X3.22-1973 AND (1600 CPI, PE), X3.39-1973

The standards X3.22-1973 and X3.39-1973 for recorded magnetic tape provide specifications for the format and recording of half inch, 9-track magnetic tape to be used for information interchange with systems and associated equipment utilizing the ASCII code, X3.4-1976. These standards define the physical requirements for magnetic tape and the bit pattern for the recording of information on 9-track tape. Seven of the nine tracks are used for data; one track is used for character parity; and one track is treated as a bit of higher order than ASCII bits (shall be a zero bit). Each of the tracks used for the recording of data are assigned a particular bit identification which corresponds to the bit assignment used by the ASCII character code standard.

COBOL 74 does not reference the recorded magnetic tape standards; however, it includes COBOL syntax and general rules governing the semantics for the COBOL statements which apply specifically to the treatment of files on magnetic

[†] Each "*" represents a tape mark.

tape. Within COBOL 74, Sequential I-O module statements such as the CLOSE REEL, CLOSE WITH NO REWIND, OPEN WITH NO REWIND and OPEN REVERSED are used to specify the actions for files stored on tape or analogous sequential media. Also the general rules associated with the READ and WRITE verbs define the actions to be taken when an end of reel is recognized during a read or write.

USING COBOL WITH RELATED STANDARDS

Programs written in compliance with the 1974 COBOL Standard language specification do not necessarily accept or produce data files that comply with the other data processing standards on many systems. Part of our work, here at the Federal COBOL Compiler Testing Service, involves the validation of COBOL compilers with respect to Federal Standard COBOL[8] (ANS, X3.23-1974 COBOL). Because we are continuously implementing the COBOL Compiler Validation System (CCVS)[9] on different computer systems, the underlying purposes for which standards are developed are of vital importance to our operations. More importantly, we test COBOL compilers and as part of those tests we validate the compiler's capability to read, process and write ASCII in accordance with the 1974 COBOL Standard.

Briefly stated, the 1974 CCVS consists of some 250 COBOL programs which collectively test a COBOL compiler's conformance to COBOL 74 language syntax and semantics. That part of the CCVS used to validate the compiler's ASCII capability consist of three COBOL programs, an ASCII-encoded punched card deck, and an ASCII-encoded magnetic tape. The ASCII validation requires three steps. In the first step, the first of the three-program set produces an ASCII-encoded magnetic tape and punched card file. The two remaining COBOL programs read and check the files produced by the first COBOL program. For the second step, the two programs which read and check the ASCII-encoded files are again executed. However, for this run they use as inputs an ASCII-encoded tape file and punched card file produced on another computer system. The purpose of this run is to validate that the compiler/computer system being tested is capable of processing ASCII (represented both on magnetic tape and punched cards) that was produced by another computer system in accordance with the appropriate American National Standards. The final step is to take the magnetic tape and punched card file produced during the validation to a different computer system for further checking. The purpose of this latter step is to determine whether the compiler/computer system being tested can also produce ASCII (represented both on magnetic tape and punched cards) which is capable of being processed on another computer system.

Since COBOL 74 is not definitive regarding labels that are used for magnetic tape, the labels used for the ASCII portion of the compiler validation are dependent on the system under test. If the system under test supports the ANSI tape label standard, then tape labels conforming to ANS X3.27-1976 are placed on the magnetic tapes. If the system under test does not support ANSI tape labels then the magnetic tapes for the validation were supplied without tape labels.

To recognize a system under test as being in full support of COBOL and associated data related standards, the system is expected to do the following:

(1) Compile and execute the audit routines (CCVS programs) that conform to the 1974 COBOL Standard (ANS X3.23-1974) without requiring any modifications to the source code,
(2) Pass all the semantic tests made by the audit routines as reflected in the execution report of each audit routine,
(3) Accept ASCII-encoded (ANS X3.4-1976) data files from tape and punched cards that conform to the Magnetic Tape Standard (ANS X3.22-1973 and ANSI X3.39-1973) and Hollerith Punched Card Code Standard (ANS X3.26-1970) respectively,
(4) Produce ASCII-encoded data files on tape and punched cards that conform to the same standards noted in 3 above, and
(5) Accept and produce magnetic tapes containing magnetic tape labels that conform to Magnetic Tape Label Standard (ANS X3.27-1976).

Very few of the compiler/computer systems validated to date were successful in meeting all of the above criteria. The type of problems that were found follow:

(1) *Date recording formats unacceptable.* One system tested resulted in an end of file condition occurring after processing the first record of a 50 record tape file. Another system could not read either a labeled or an unlabeled magnetic tape. However, this same system, for some unexplained reason, was able to create a tape that, when read on another computer system, proved to be correct according to applicable standards.
(2) *Data code not read or recorded correctly.* One system tested produced records on tape which included an additional character located in the first character position of each record thus causing the contents of the record to be offset by one character. Another system did not provide proper character translation of all ASCII characters when reading punched cards.
(3) *Incompatible Tape labels.* Some systems tested simply did not support ANSI tape labels while others required the presence of optional labels HDR2 and EOF2.
(4) *Non-standard COBOL syntax.* One system tested required that the nonstandard clause "RECORDING MODE IS STANDARD-ASCII" be added to the SELECT Entry for all tape files. (The CODE-SET clause in the File Description Entry should have accomplished this function). One system, oddly enough,

could properly read and write ASCII on punched cards only when the CODE-SET clause was removed from the program.

A number of special considerations were necessary in order to process ASCII. Some implementations required that specific hardware models of magnetic tape and card readers/punches be used. Others required that specific options be invoked through the job control language, and still others required particular system generation parameters be used when the operating system is generated.

IMPACT OF COBOL 74 REVISION WITH REGARD TO OTHER STANDARDS

In looking to the future to see how the revision to COBOL 74 might affect the interfaces of COBOL with related standards, it appears that there will be little or no impact. In a sense this is unfortunate because the COBOL standard could give more guidance as to the type of file labels that are to be used. The only feature for defining file labels within COBOL is via the LABEL RECORD STANDARD clause, which in COBOL 74 merely requires that labels will be present and conform to the implementor's label specifications.

The difficulty most often found in transporting magnetic tape files between computer systems, next to improper implementation of standards related to COBOL, was the tape label. Many COBOL compiler/computer systems tested did not support ANSI magnetic tape labels. If the COBOL interface mechanisms for ANSI magnetic tape labels were similar in detail to the interfaces now present for ASCII, COBOL would have a direct link to another American National Standard.

CONCLUSION

One thing became quite apparent while testing the interfaces to ASCII and other standards related to COBOL. Even when strict adherence to American National Standards is maintained, COBOL programs and associated data files are not easily interchangeable among computer systems. This, in part, is due to improper implementation of ADP Standards by the parties involved in the interchange. Another factor is that the goals and objectives of X3 do not emphasize the compatibility of related standards or encourage interaction of related standards.

REFERENCES

1. *American National Standard Programming Language COBOL, X3.23-1974,* American National Standards Institute Incorporated, New York, 1974
2. *American National Standard Code for Information Interchange, X3.4-1976,* American National Standards Institute Incorporated, New York, 1976
3. *American National Standard Hollerith Punched Card Code, X3.26-1970,* American National Standards Institute Incorporated, New York, 1970
4. *American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI), X3.22-1973,* American National Standards Institute Incorporated, New York, 1973
5. *American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1976,* American National Standards Institute Incorporated, New York, 1976
6. Hill, Marjorie F., *The World of EDP Standards,* Control Data Corporation, Tech Memo TM4, 1972
7. *American National Standard Recorded Magnetic Tape for Information Interchange (1600 CPI, Phase Encoded), X3.39-1973,* American National Standards Institute Incorporated, New York, 1973
8. *Federal Information Processing Standards,* Publication 21-1 COBOL, U. S. Government Printing Office, Washington, D.C., 1975
9. *CCVS Users Guide Version 3.0,* available from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, Virginia 22151

Ciphertext/plaintext and ciphertext/key dependence vs number of rounds for the data encryption standard

by CARL H. MEYER
International Business Machines Corporation
Kingston, New York

INTRODUCTION

One property of the Data Encryption Standard (DES) is that each bit of ciphertext is a complicated function of all plaintext bits and all key bits. A method is developed which evaluates how fast this dependence (defined as intersymbol dependence) builds up as a function of repeated mathematical operations called "rounds." It is shown that the minimum number of rounds to achieve intersymbol dependence for plaintext as well as key is five.

Cryptography deals with the methods involved in preparing cryptograms—messages or writings intended to be incomprehensible to all except those who legitimately possess the means to recover the original information. The designer of a cryptographic system, or cryptosystem, is a cryptographer.

The opponent or antagonist of a cryptosystem is a cryptanalyst. Cryptanalysis is concerned with techniques used to penetrate communications and recover the original information by means other than those available to the legitimate recipient.

Cryptology is the science of disguised or secret communications. It embraces both cryptography and cryptanalysis.

The basic challenge in cryptography is to devise a method that transforms messages (known as plaintext) into cryptograms (known as ciphertext) in a cryptographically secure way—that is, the method must withstand intense efforts of cryptanalysis. Plaintext can be protected by either of two techniques: it can be encoded using a code system, or it can be enciphered (encrypted) using a cipher system.

Code systems require a code book or dictionary that relates the words, phrases, and sentences of the vocabulary (the plaintext) to its equivalent code group (the ciphertext), and vice versa. The number of plaintext messages that can be encoded depends on the number of combinations of phrases that can be obtained from the code book. Although that number may be large, not every combination or pattern of bits can be encoded. Hence the versatility and usefulness of code systems is limited, especially in computer applications, which would have to be programmed to handle all such exceptions.

Cipher systems require two basic elements: a set of rules or steps—that is, an algorithm—constituting the basic cryptographic procedure, which is constant in character and agreed upon in advance, and a specific cryptographic key, selected from a large set of possible keys, which is known only by the communicators.

A cryptographic algorithm can be represented as an extremely large number of possible mathematical procedures called transformations. Each transformation defines how sequences of intelligible data (representing a message) are changed into sequences of apparently random noise (gibberish) that are unintelligible to humans or machines. The cryptographic key is a secretly held sequence of numbers of characters, usually very short, which identifies the transformation to be used.

To be useful, the algorithm should have, for each of its transformations, a unique inverse operation that changes the gibberish back into intelligible data. (This process is called decipherment or decryptment.) However, this can be done only if one has exact knowledge of the key that was used in the initial encipherment (transformation). A cryptographic algorithm will, therefore, provide data security between two nodes of a data processing system if those two nodes have the algorithm installed (in hardware or software), and if both nodes have exact knowledge of the key.

It is important to note that for good data security, only the key need be kept secret. The details of the algorithm are assumed to be known to everyone. The cryptographic key, therefore, is often considered analogous to the secretly held combination number of a safe.

It is always possible to construct an unbreakable system if the number of characters in the key is equal to or greater than the number of characters of plaintext to be enciphered.[1] It is required, however, that the key be randomly selected and used only once. This approach is impractical in data processing systems because of the large amount of message traffic. In a practical system, the key must be of fixed length, relatively short, and capable of repeated use.

Basically there are three general classes of ciphers: transposition ciphers, substitution ciphers, and combinations of these called product ciphers. A transposition cipher involves

the rearrangement or permutation of the plaintext symbols without change in their identity. A substitution cipher involves the replacement of plaintext symbols by one or more symbols without changing their sequence. Cryptographic research conducted during World War II showed that strong encryption systems could be obtained using alternate steps of substitution and transposition, resulting in a product cipher.[1]

Further research into the development of strong product ciphers was not undertaken in the private sector until the late 1960's. During the period from 1968 to 1975, a cryptographic procedure consisting of 16 alternate steps (or rounds) of key-controlled substitution and fixed permutation, based on work done by Horst Feistel, was developed at IBM.[2] This algorithm was accepted as a standard by the National Bureau of Standards on July 15, 1977, and is known as the Data Encryption Standard (DES).[3]

Next in importance to having a strong algorithm is the incorporation of the cryptographic device into the system. In a weak implementation, by manipulating certain cryptographic functions, it may be possible for an opponent to obtain the desired information. For an example of a strong system implementation of the DES, see Reference 4 from which this introductory material was taken.

Furthermore, if the key required for decrypting data is lost or unknown, the data cannot be recovered from the ciphertext. It is just as difficult for the authorized user to decrypt these data when the key is unknown as it is for the opponent.

The Data Encryption Standard's principle of operation (without initial and final permutation on input and output, respectively) is shown in Figure 1. For strength, the DES relies on (1) the complexity of the function g, which incorporates substitution as well as transposition[2] (or permutation), and (2) exercising the function g a sufficient number of times (defined as rounds).

After 16 rounds are employed by the DES, the 64 bits of plaintext are transformed into 64 bits of ciphertext under the control of a 56 bit key. During each round a subset of the key [defined as $K(i)$ for round i; $i=1,2,\ldots,16$] is used.[3] The input to round i, $X(i-1)$, is expressed as the concatenation of two 32 bit quantities:

$L(i-1)$ represents the 32 input bits to the left half of the one round cipher operation (see Figure 1) and $R(i-1)$ the 32 input bits to the right half, i.e.,

$$L(i-1)=l_1(i-1), l_2(i-1), \ldots, l_{32}(i-1)$$

$$R(i-1)=r_1(i-1), r_2(i-1), \ldots, r_{32}(i-1)$$

Hence the input to round (i) is defined as

$$X(i)=L(i), R(i).$$

whereas the output from round (i) is defined as

$$X(i-1)=L(i-1), R(i-1),$$

One property of the DES is that each bit of the produced ciphertext is a complicated function of all 64 plaintext bits and all 56 key bits.

To give a representative demonstration of this property let X represent a 64-bit block of input data (plaintext), Y a 64-bit block of output data (ciphertext), and K a 64-bit cryptographic key (eight bits may be used for parity checking, hence are not used by the algorithm itself). The notation $f_K(X)=Y$ means that the encipherment (f) of X under key K is equal to Y, and the notation $f^{-1}{}_K(Y)=X$ means that the decipherment (f^{-1}) of Y under key K is equal to X.

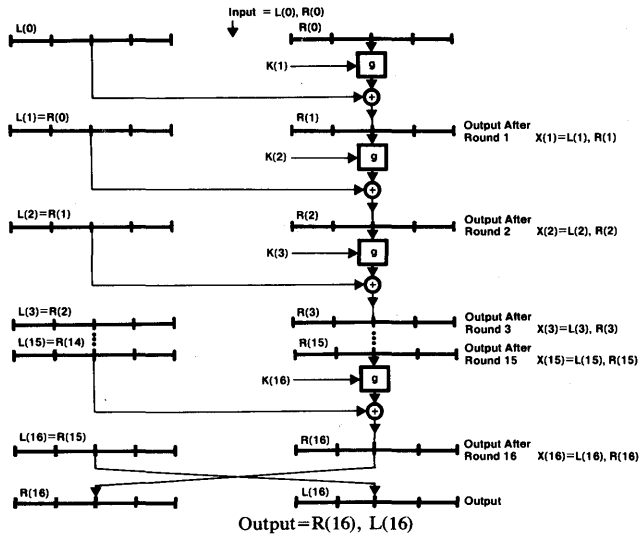


Figure 1—Basic block cipher design used by the data encryption standard

Enciphering the plaintext

$$X=1000000000000001$$

with a key

$$K=3001010101010101$$

results in

$$Y=f_K(X)=958E6E627A05557B$$

where hexadecimal notation is used.

If the same key is used to decipher this quantity, X is recovered.

However, by letting

$$Y^*=858E6E627A05557B$$

instead of Y be deciphered with K, where Y^* differs only in 1 bit from Y, the result is

$$f^{-1}{}_K(Y^*)=8D4893C2966CC21$$

which shows the drastic effect of only a single bit ciphertext change on the decipherment process and, hence, indicates how strong a ciphertext bit depends on all plaintext bits.

If, on the other hand, a key

$$K^*=1001010101010101$$

is used to decipher Y, where the 56 independent key bits of K^* differ from K in only 1 bit, the result is

$$f^{-1}{}_K{}^*(y)=6D4B945376725395$$

which shows the effect of only a one bit key change on the decipherment procedure and, in this case, indicates how strong a ciphertext bit depends on all key bits.

The dependence of each ciphertext bit on all plaintext and key bits is defined here as "intersymbol dependence."

There are many applications which can take advantage of this.[5] Two applications, which are demonstrated below, effectively expand the block size of the DES by using chaining methods. When block encryption is used, each 64-bit block of data is enciphered separately. In chained block encryption on the other hand, the encipherment of each block is also made dependent on prior information (plaintext, ciphertext, or the like) that is available when the block is enciphered. Two important chained block encryption techniques are ciphertext feedback and plaintext-ciphertext feedback, as defined below.

Let $X_1, X_2, \ldots, X_n$ denote blocks of plaintext to be chained using key K, let Y_0 be a nonsecret quantity defined initializing vector, and Let $Y_1, Y_2, \ldots, Y_n$ denote the blocks of ciphertext produced. When ciphertext feedback is used, the following relationship holds:

$$Y_i = f_K(X_i \oplus Y_{i-1}) \text{ for } i \geq 1$$

where $\oplus$ represents module 2 addition. When plaintext-ciphertext feedback is used, the following relationships hold:

$$Y_1 = f_K(X_1 \oplus Y_0)$$

and

$$Y_i = f_K(X_i \oplus Y_{i-1} \oplus X_{i-1}) \text{ for } i \geq 1.$$

With blockchaining a ciphertext bit in block i therefore depends not only on all plaintext bits in block i but also on the plaintext bits of block 1 through $(i-1)$. Thus chained block encryption (or block chaining) can be used to extend the intersymbol dependence between ciphertext and plaintext.

Plaintext-ciphertext feedback has the additional property of error propagation. Corruption of a single bit of ciphertext will cause each subsequent bit of recovered plaintext to be in error with probability 0.5. By appending a known pattern of bits to the end of the plaintext prior to encryption, and comparing that value to the value recovered, the error propagation feature can be used for checking the true context of a message. Chaining techniques are useful for encrypting data, and the block cipher (with no chaining) is useful for key transformation operations.[5]

In the analysis to follow, a method is developed which shows how fast the intersymbol dependence builds up as a function of the number of rounds. The approach evaluates whether or not an output bit depends upon an input bit. Although the degree of complexity is not measured, a distinction is made between three kinds of functional relationships. To discuss these relations, details of g (the kernel of the DES cryptographic approach) are illustrated in Figure 2.

The right half of the input to round (i) is expanded from 32 bits to 48 bits and is denoted by $E[R(i-1]$, using the E Bit-Selection Table.[3] The result is:

$$\begin{aligned} E[R(i-1] = {} & r_{32}, r_1, r_2, r_3, r_4, r_5, r_4, r_5, r_6, r_7, r_8, r_9, r_8, r_9, r_{10}, r_{11}, \\ & r_{12}, r_{13}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{16}, r_{17}, r_{18}, r_{19}, \\ & r_{20}, r_{21}, r_{20}, r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{24}, r_{25}, r_{26}, r_{27}, \\ & r_{28}, r_{29}, r_{28}, r_{29}, r_{30}, r_{31}, r_{32}, r_1 \end{aligned} \quad (1)$$

where the parentheses have been dropped for convenience of representation. The purpose of using the E expansion is to achieve a dependence of each ciphertext bit on all plaintext bits in as few rounds as possible.

Next $E[R(i-1)]$ is added modulo-2 to $K(i)$, resulting in a 48 element vector, (A).

$$A = E[R(i-1)] \oplus K(i)$$

$$= a_1, a_2, \ldots, a_{48}.$$

[Although (A) is different for each round and should therefore be indexed by (i), to avoid unnecessary details, it is not done here.]

In Figure 2 the elements of vector (A) are used as arguments in the substitution operations (S-boxes) S_1 through S_8.[3] Each S-box is described as a matrix of four rows (labeled 00, 01, 10, 11) and 16 columns (labeled 0000, 0001, . . . , 1111).

Each S-box can now be represented by four substitution functions, i.e., S_i^{00}, S_i^{01}, S_i^{10}, S_i^{11} for S-box S_i where the superscript identifies the row of the matrix, each mapping four input bits (which determine the column of the matrix) to four output bits given by the element of the matrix.

The first and last input of the six entries to the DES substitution box S_i in Figure 2 determine which of the four substitution functions in S_i are selected. Because these bits are derived from input bits (i.e., message bits if the input is interpreted as a message) as well as key bits, the selection of substitution functions not only depends on the key but also on the input data. Thus the E expansion, in effect, introduces an autoclave or self keying feature. Since (i) ranges from one through eight, there are 32 functions that are obtained with the eight S-boxes.

With a_1 through a_6 being the inputs to the first S-box,

$$S_1^{a_1, a_6}(a_2, a_3, a_4, a_5)$$

represent the first four bits of (B), i.e., bits b_1 through b_4.

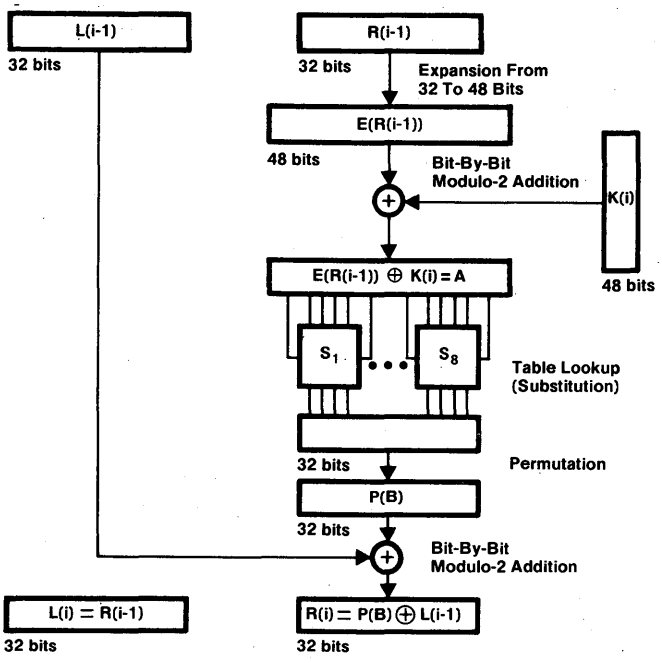


Figure 2—Details of (g) enciphering function

The last four bits, from S_8, i.e., b_{29} through b_{32} are given by

$$S_8{}^{a_{43},a_{48}}(a_{44}, a_{45}, a_{46}, a_{47}).$$

Knowledge of the 48 bits represented by vector (A) therefore allows the 32 bits represented by vector (B) to be determined. By permuting the elements of B, $P(B)$ is obtained as follows.[3]

$$P(B)=b_{16}, b_7, b_{20}, b_{21}, b_{29}, b_{12}, b_{28}, b_{17}, b_1, b_{15}, b_{23}, b_{26}, b_5, b_{18}, b_{31}, b_{10}, b_2, b_8, b_{24}, b_{14}, b_{32}, b_{27}, b_3, b_9, b_{19}, b_{13}, b_{30}, b_6, b_{22}, b_{11}, b_4, b_{25} \quad (2)$$

The output of round (i) thus becomes

$$L(i), R(i)=R(i-1), P(B)\oplus L(i-1)$$

Furthermore,

$$L(i)=R(i-1) \quad (3)$$

and

$$R(i)=L(i-1)\oplus g(K(i), R(i-1)) \quad (4)$$

where the 48 bits of $K(i)$ are a subset of the inputted key as previously discussed.

In the following two analyses, the dependencies of output $X(i)$ on plaintext $X(0)$ and on the key are treated separately.

INTERDEPENDENCE BETWEEN CIPHERTEXT AND PLAINTEXT

To investigate the functional relationship between the input to the $(i+1)$-th round (which is equal to the output of the (i)-th round, $i=1, 2, 3, \ldots, 16$), and the input to the first round, a matrix is defined. Consisting of 64 rows and 64 columns and referred to as $G_{i,j}$, its element $a_{l,m}$ for row l and column m shows the type of relationship which exists between the l-th bit of $X(i)$ and the m-th bit of $X(j)$. In particular, $a_{l,m}$ is "blank" if a dependence does not exist between $x_l(i)$ and $x_m(j)$. If there is a dependence via message bits only, $a_{l,m}$ is set to "x". If the dependence is via autoclave, $a_{l,m}$ is set to "$-$". If message bits as well as autoclave bits influence the output, $a_{l,m}$ is set to "*".

In a well-designed system, an output bit depends on more and more input bits as the number of rounds increases. These input bits come into play via message (x) dependence, autoclave $(-)$ dependence, or both (*). The design goal is to have each output bit depend on each input bit after only a few rounds, with autoclave as well as message dependence being achieved.

It is advantageous to partition matrix $G_{i,j}$ into four submatrices of 32 rows and 32 columns each, i.e.,

$$G_{i,j}=\begin{bmatrix} G_{i,j}^{(L,L)} & G_{i,j}^{(L,R)} \\ G_{i,j}^{(R,L)} & G_{i,j}^{(R,R)} \end{bmatrix}$$

Using the definition of $G_{i,j}$, the submatrices' elements express the relationships as shown in Table I.

Let the relationships between the output and the input of one round, i.e., $G_{i,i-1}$ be evaluated first. The dependence of the output bits from the substitution operation (vector (B) in Figure 2) on the input bits $R(i-1)$, explained below, is shown in Figure 3. [The numbers identifying the row and column of the matrix also refer to the index of the element of the appropriate vector, i.e., (B) vs $R(i-1)$, respectively, in Figure 3. Furthermore, note that (B) vs $R(i-1)$ is identical to $R(i)$ vs $R(i-1)$ if the permutation is not present].

TABLE I.—Functional Relationships

Submatrix	Relationship expressed in submatrix
$G_{i,j}^{(L,L)}$	$L(i)$ vs $L(j)$
$G_{i,j}^{(L,R)}$	$L(i)$ vs $R(j)$
$G_{i,j}^{(R,L)}$	$R(i)$ vs $L(j)$
$G_{i,j}^{(R,R)}$	$R(i)$ vs $R(j)$

The selection of the substitution function (S-function) in the first S-box, S_1, depends on the 32nd and fifth bit of $R(i-1)$. This follows from the fact that (1) the first and sixth input bits to S_1 select the S-function, and (2) the first and sixth bits of the expanded version of $R(i-1)$, which is determined by $E[R(i-1)]$, is equal to $r_{32}(i-1)$ and $r_5(i-1)$, respectively, according to Eq. 1. Therefore, all output bits from S_1, i.e., b_1 through b_4, depend on $r_{32}(i-1)$ and $r_5(i-1)$ via autoclave. However, they depend on $r_1(i-1)$ through $r_4(i-1)$ via message. $R(i-1)$ is considered to be the input message in this case. Hence, the entries in rows 1 through 4, columns 32 and 5 in Figure 3 are equal to "$-$". The corresponding entries in columns 1 through 4 are equal to "x". All other entries of rows 1 through 4 are blank since b_1 through b_4 only depend on six input bits. Repeating this analysis for S_2 through S_8 results in the matrix of Figure 3.

Taking permutation into account, the matrix rows have to be rearranged according to the permutation schedule given by $P(B)$ in Eq. 2. The result is shown in Figure 4 and rep-

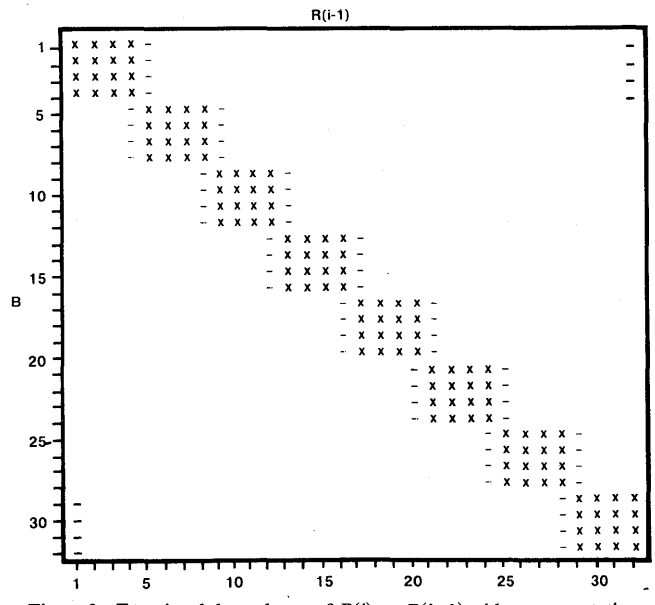


Figure 3—Functional dependence of $R(i)$ on $R(i-1)$ without permutation

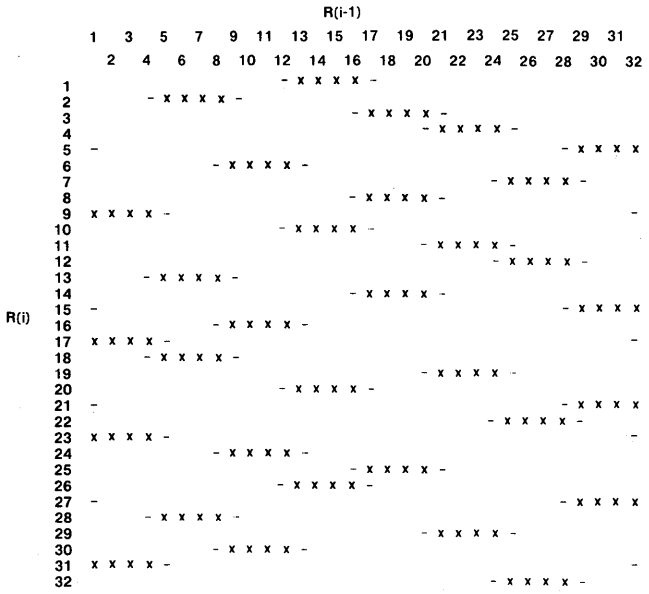


Figure 4—Functional dependence of $R(i)$ on $R(i-1)$, matrix $G_{i,i-1}^{(R,R)}$

resents the type of relationship that exists between $R(i)$ and $R(i-1)$; thus, $G_{i,i-1}^{(R,R)}$ is obtained (see Table I).

From $L(i)=R(i-1)$, see Eq. 3, $L(i)$ does not depend on $L(i-1)$ but has a linear dependence on $R(i-1)$. Hence the elements of $G_{i,i-1}^{(L,L)}$ are blank since they express the relationship between $L(i)$ and $L(i-1)$.

The relationship between $L(i)$ and $R(i-1)$ is expressed by the elements of $G_{i,i-1}^{(L,R)}$. Due to the linear relationship between $L(i)$ and $R(i-1)$, as discussed before, the elements located on the diagonal are set to "x" as shown in Figure 5. Eq. 4 shows the relationship of $R(i)$, $L(i-1)$, $K(i)$, and $R(i-1)$. Since the dependence on $K(i)$ is not of interest here, the expression

$$R(i)=L(i-1)+h[R(i-1)] \tag{5}$$

shall be used hereafter. The relationship, therefore, between $R(i)$ and $L(i-1)$ is also linear and, hence, the diagonal elements of $G_{i,i-1}^{(R,L)}$, which relate $R(i)$ and $L(i-1)$, are set to "x", whereas the remaining elements are blank, as shown in Figure 5.

Having evaluated $G_{i,i-1}^{(L,L)}$ through $G_{i,i-1}^{(R,R)}$, the matrix $G_{i,i-1}$ is completely defined and is shown in Figure 5.

Let $G_{i+1,i-1}$ be expressed next in terms of $G_{i,i-1}$. From Eq. 3 the relation $L(i+1)=R(i)$ can be established. Thus the functional dependence of $L(i+1)$ on $X(i-1)$ is identical to the functional dependence of $R(i)$ on $X(i-1)$. Since the dependence of $L(i+1)$ on $X(i-1)=[L(i-1), R(i-1)]$ is given by $G_{i+1,i-1}^{(L,L)}$, $G_{i+1,i-1}^{(L,R)}$, and the dependence of $R(i)$ on $X(i-1)$ is given by $G_{i,i-1}^{(R,L)}$, $G_{i,i-1}^{(R,R)}$, according to Table I, it follows that

$$G_{i+1,i-1}^{(L,L)}=G_{i,i-1}^{(R,L)} \tag{6}$$

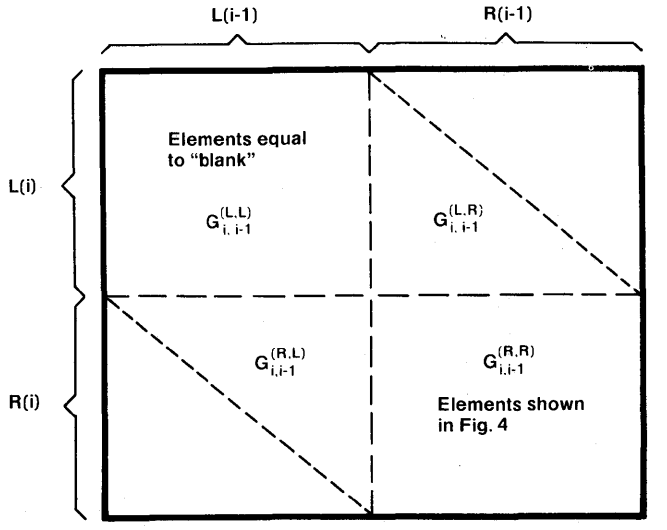
and

$$G_{i+1,i-1}^{(L,R)}=G_{i,i-1}^{(R,R)} \tag{7}$$

Before showing how $G_{i+1,i-1}^{(L,L)}$ and $G_{i+1,i-1}^{(R,R)}$ may be obtained, a more general case is considered, i.e., the evaluation of $G_{j+1,i-1}^{(R,L)}$ and $G_{j+1,i-1}^{(R,R)}$ ($j\geq i$) from $G_{j,i-1}$. (Note that for $j=i$ the above stated special case is obtained).

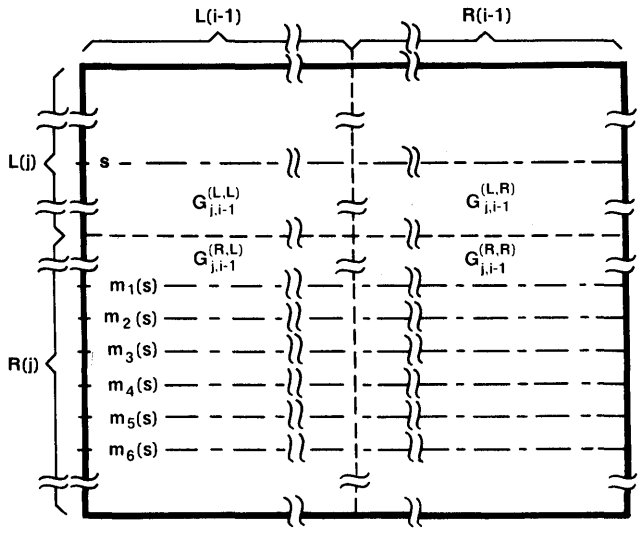
The elements of row s of $G_{j+1,i-1}^{(R,L)}$ give the relationship between bit s of $R(j+1)$, i.e., $r_s(j+1)$ and $L(i-1)$, whereas the elements of row s of $G_{j+1,i-1}^{(R,R)}$ give the relationship between bit s of $r(j+1)$ and $R(i-1)$. To evaluate these elements the relationship between $r_s(j+1)$ and $X(j)=[L(j), R(j)]$ is obtained first by using Figures 4 and 5. Bit s of $R(j+1)$ linearly depends on $L(j)$ as shown in Figure 5, i.e., on bit s of $L(j)$ via message dependence (x). From Figure 4 it follows that bit s of $R(j+1)$ depends on $R(j)$ via -$xxxx$-, i.e., bit s depends on two bits of $R(j)$ via autoclave and on four bits of $R(j)$ via a message relationship. Let the column locations of these entries in $G_{i,i-1}^{(R,R)}$ (equal to $G_{j+1,j}^{(R,R)}$ if $i=j+1$) be designated as $m_1(s)$ through $m_6(s)$. For example, using Figure 4, the following is obtained for $s=4$: $m_1(s)=20$, $m_2(s)=21$, $m_3(s)=22$, $m_4(s)=23$, $m_5(s)=24$, and $m_6(s)=25$. Since the dependence of $X(j)$ on $X(i-1)$ is given by $G_{j,i-1}$ and the dependence of bit s of $R(j+1)$ on $X(j)$ is known, the relationship between $R(j+1)$ and $X(i-1)$, given by rows 32 to 64 of matrix $G_{j+1,i-1}$ can be determined by properly combining elements of row s, $m_1(s)+32$, . . . , $m_6(s)+32$ of $G_{j,i-1}$. This method is indicated graphically in Figure 6.

The question still to be answered is, "What rule should be used to combine rows of matrix $G_{j,i-1}$ so that the dependence of $x_{s+32}(j+1)$ on $X(i-1)$ can be evaluated?" Since the main objective of this analysis is to determine how fast the functional dependence between output and input builds up it would be sufficient to simply indicate if a functional relationship between an output bit and an input bit exists or not. However, to gain more insight on the influence of au-



Elements of $G_{i,i-1}^{(L,R)}$ and $G_{i,i-1}^{(R,L)}$ located on the indicated diagonal are equal to "x" whereas the remaining elements are equal to blank.

Figure 5—Functional relationship between $X(i)=L(i)$, $R(i)$ and $X(i-1)=L(i-1)$, $R(i-1)$, matrix $G_{i,i-1}$



Note: X(j + 1) = [L(j + 1), R(j + 1)]vs X(i-1) is obtained by combining the elements of row s, $m_1(s)$ through $m_6(s)$, where $m_1(s)$ through $m_6(s)$ are the columns in which the elements in row s of $G^{(R,R)}_{i,i-1}$ occur. ($1 \leqslant s \leqslant 32$.)
$G^{(R,R)}_{i,i-1}$ is defined in Figure 4

Figure 6—Evaluation of functional dependence of $X(j+1)$ vs $X(i-1)$ from $X(j)$ vs. $X(i-1)$

toclave, a rule was devised which highlights the autoclave influence. Thus all elements of $G_{j,i-1}$ for rows $m_1(s)+32$ and $m_6(s)+32$ are set to "−", indicating an autoclave relationship. Note that the elements of row $s+32$ of $G_{j+1,i-1}$ describe the dependence of $x_{s+32}(j+1)$ on $X(i-1)$. Since $x_{s+32}(j+1)$ depends on $x_{m_1(s)+32}(j)$ and $x_{m_6(s)+32}(j)$ via an autoclave relationship, the dependence of $x_{m_1(s)+32}(j)$ and $x_{m_6(s)+32}(j)$ on $X(i-1)$ is changed from "−", "x" or "*" to an autoclave dependence "−". By adopting this rule, previous history, as far as the autoclave entry to an S-box is concerned; is eliminated, thereby increasing the emphasis on the last autoclave dependence.

In order to take into account previous history for the entries to an S-box not associated with autoclave, the elements of $G_{j,i-1}$ are left unchanged for rows s, $m_2(s)+32$, $m_3(s)+32$, $m_4(s)+32$ and $m_5(s)+32$.

In combining the proper rows of $G_{j,i-1}$ one or more entries can occur in the same column. Only if entries are either all "x" or "−" will the final dependence be correspondingly set. If mixed entries ("x" and "−") occur, the final dependence is set to "*". If entry "*" occurs at least once, then the final dependence is set to "*", regardless of what the other entries are.

Procedure summary

To generate the elements of row $(s+32)$ of matrix $G_{j+1,i-1}$, for $1 \leq s \leq 32$ determine $m_1(s)$ through $m_6(s)$ from the given matrix $G_{i,i-1}{}^{(R,R)}$. They are the columns in which the nonzero (i.e., non-blank) elements in row s of $G_{i,i-1}{}^{(R,R)}$ are located (Figure 4 gives these entries). Take matrix $G_{j,i-1}$ and select row s, $m_1(s)+32$ through $m_6(s)+32$. The entries in row $(s+32)$ of matrix $G_{j+1,i-1}$ are now obtained by combining the elements in the indicated rows using the established rules.

As a corollary, the following can be stated:

Elements in row s of $G_{j+1,i-1}{}^{(R,L)}$ are obtained by combining row s of $G_{j,i-1}{}^{(L,L)}$ with rows $m_1(s)$ through $m_6(s)$ of $G_{j,i-1}{}^{(R,L)}$. Elements in row s of $G_{j+1,i-1}{}^{(R,R)}$ are obtained by combining row s of $G_{j,i-1}{}^{(L,R)}$ with rows $m_1(s)$ through $m_6(s)$ *of* $G_{j,i-1}{}^{(R,R)}$.

Let the special case $j=i$ be considered now, i.e., let $G_{i+1,i-1}$ be evaluated from $G_{i,i-1}$. Using the stated rules, the submatrix $G_{i+1,i-1}{}^{(R,R)}$ can be derived. The result is shown in Figure 7.

Furthermore, it can be shown that

$$G_{i+1,i-1}{}^{(R,L)} = G_{i,i-1}{}^{(R,R)}$$

using an approach similar to the one applied in the proof by induction shown below and hence (from Eq. 7)

$$G_{i+1,i-1}{}^{(R,L)} = G_{i+1,i-1}{}^{(L,R)} = G_{i,i-1}{}^{(R,R)} \tag{8}$$

Using also Eq. 6 the evaluation of $G_{i+1,i-1}$ in terms of $G_{i,i-1}$ can be completed.

The proof by induction is used next to show that the following general rule (for $j \geq i$) holds:

$$G_{j+1,i-1}{}^{(L,L)} = G_{j,i-1}{}^{(R,L)} \tag{9}$$

$$G_{j+1,i-1}{}^{(L,R)} = G_{j,i-1}{}^{(R,R)} \tag{10}$$

$$G_{j+1,i-1}{}^{(R,L)} = G_{j+1,i-1}{}^{(L,R)} = G_{j,i-1}{}^{(R,R)} \tag{11}$$

The first stage of the proof was already completed by showing that these statements are correct for the smallest value of j, $j=i$. (See Eqs. 6, 7 and 8.)

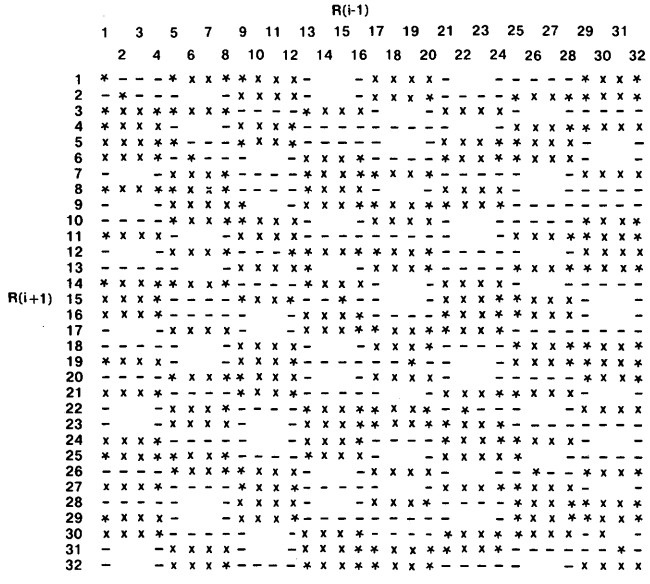


Figure 7—Functional dependence of $R(i+1)$ on $R(i-1)$, matrix $G_{i+1,i-1}{}^{(R,R)}$

From the relationship (Eq. 3)

$$L(j+1)=R(j),$$

it can be determined that $L(j+1)$ depends on $X(i-1)$ in the same way $R(j)$ depends on $X(i-1)$. Thus,

$$G_{j+1,i-1}{}^{(L,L)}=G_{j,i-1}{}^{(R,L)}$$

$$G_{j+1,i-1}{}^{(L,R)}=G_{j,i-1}{}^{(R,R)}$$

which proves Eq. 9 and 10.

To prove Eq. 11 the results stated in the corollary are used as follows:

$G_{j+1,i-1}{}^{(R,L)}$ is obtained by properly combining the elements of $G_{j,i-1}{}^{(L,L)}$ (row s) and $G_{j,i-1}{}^{(R,L)}$ (rows $m_1(s)$ through $m_6(s)$). But $G_{j,i-1}{}^{(L,L)}=G_{j-1,i-1}{}^{(R,L)}$, $G_{j,i-1}{}^{(R,L)}=G_{j-1,i-1}{}^{(R,R)}$, if the stated rule holds.

$G_{j+1,i-1}{}^{(L,R)}=G_{j,i-1}{}^{(R,R)}$ is obtained by properly combining the elements of $G_{j-1,i-1}{}^{(L,R)}$ (row s) and $G_{j-1,i-1}{}^{(R,R)}$ (rows $m_1(s)$ through $m_6(s)$). But $G_{j-1,i-1}{}^{(L,R)}=G_{j-1,i-1}{}^{(R,L)}$ if the stated rule holds.

Hence, $G_{j+1,i-1}{}^{(R,L)}$ is obtained in an identical way as $G_{j,i-1}{}^{(R,R)}$ and, likewise, $G_{j+1,i-1}{}^{(L,R)}$ is obtained. Therefore, $G_{j+1,i-1}{}^{(R,L)}=G_{j+1,i-1}{}^{(L,R)}$ which completes the proof (see Figure 8 for a graphical representation).

Minimum required number of rounds to achieve ciphertext/plaintext intersymbol dependence

For each output bit to depend on all plaintext bits after round (j), no element of $G_{j,o}$ can be blank. When this condition is satisfied, it follows from the developed relationships between $G_{j,o}$ and $G_{j-1,o}$, $G_{j-2,o}$ that (1) no elements of $G_{j-1,o}{}^{(L,R)}=G_{j-1,o}{}^{(R,L)}$ can be blank and (2) no elements of $G_{j-2,o}{}^{(R,R)}$ can be blank.

By assuming that all row entries of the matrices are independent, an approximate result for the minimum number of rounds can be obtained, as shown below. (In actuality the elements of eight sets of four rows each of $G_{i,i-1}{}^{(R,R)}$ are highly correlated. Therefore, the buildup of intersymbol dependence will be slower than predicted by the approximation.)

Since $G_{1,0}{}^{(R,R)}$ has six entries (Figure 4) in each row and $G_{1,0}{}^{(L,R)}$ has one entry in each row (Figure 5), then $G_{2,0}{}^{(R,R)}$ can have at most 1+6·6 or 37 entries. (Note that $G_{2,0}{}^{(R,R)}$ is obtained from $G_{1,0}{}^{(L,R)}$ and $G_{1,0}{}^{(R,R)}$.) There are, however, only 32 columns in $G_{2,0}{}^{(R,R)}$. Therefore, it is possible that none of the elements of $G_{2,0}{}^{(R,R)}$ are blank. Thus, it is possible to fill all 32^2 entries of $G_{2,0}{}^{(R,R)}$, $G_{3,0}{}^{(L,R)}=G_{3,0}{}^{(R,L)}$, $G_{4,0}{}^{(R,R)}$ after rounds 2, 3 and 4 respectively. Hence, all 64^2 entries of $G_{4,0}$ could be filled after four rounds.

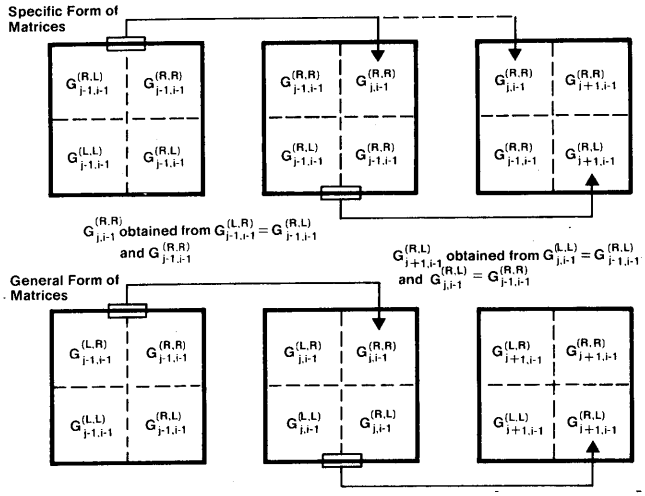


Figure 8—Graphical presentation of proof that $G_{j+1,i-1}{}^{(R,L)}=G_{j+1,i-1}{}^{(L,R)}$

From the evaluation of $G_{i+1,i-1}{}^{(R,R)}$ (Figure 7) there are 28 non-blank elements in each row, except for row 30 which contains 31 elements. It can also be shown that all entries of $G_{i+2,i-1}{}^{(R,R)}$ are equal to "*". Thus, after Round 4 each output bit of $L(4)$, except bit 30, depends on 28 bits of $L(0)$ and all 32 bits of $R(0)$. Bit 30 of $L(4)$ depends on 29 bits of $L(0)$ and all 32 bits of $R(0)$. Each bit of $R(4)$, on the other hand, respectively depends on all 32 plaintext bits of $L(0)$ and $R(0)$.

Let the degree of intersymbol dependence be measured

TABLE II.—Ciphertext/Plaintext Intersymbol Dependence

Round j	Output/Input Relation				
	$L(j)$ vs $L(O)$	$L(j)$ vs $R(0)$	$R(j)$ vs $L(0)$	$R(j)$ vs $R(0)$	$X(j)$ vs $X(0)$
1	0.00	3.13	3.13	18.75	6.26
2	3.13	18.75	18.75	88.18	32.20
3	18.75	88.18	88.18	100.00	73.78
4	88.18	100.00	100.00	100.00	97.05
5	100.00	100.00	100.00	100.00	100.00

NOTE: Table entries express the degree of intersymbol dependence, i.e., the percentage of non-blank elements in the appropriate relation.

by a factor, ξ, which is obtained by evaluating the percentage of non-blank elements. Respectively, for $G_{1,0}^{(L,L)}$ through $G_{1,0}^{(R,R)}$ (Figures 4 and 5), ξ is 0; $100 \cdot 32/32^2 = 3.125$; 3.125; and $100 \cdot 32 \cdot 6/32^2 = 18.75$ percent. With the total $G_{1,0}$, ξ is equal to 1/4 (0+3.125+3.125+18.75) or 6.25 percent. To obtain ξ for $G_{2,0}^{(R,R)}$, either Figure 7 or the previously quoted results can be used to arrive at $100 \cdot (32^2(31 \cdot 4 - 3))/32^2$ or 88.18 percent. Hence, using the rules obtaining $G_{3,0}$ from $G_{1,0}$, ξ associated with $G_{2,0}^{(L,L)}$ through $G_{2,0}^{(R,R)}$ is 3.13 percent, 18.75 percent, 18.75 percent, 88.18 percent, respectively, whereas ξ for $G_{2,0}$ is equal to 1/4 (3.125+18.75+88.18) or 32.20 percent. Because none of the elements of $G_{3,0}^{(R,R)}$ are blank (they are actually all "*") ξ, as associated with $G_{3,0}^{(R,R)}$, is equal to 100 percent. The results are summarized in Table II.

INTERDEPENDENCE BETWEEN CIPHERTEXT AND KEY

The basic ideas developed in the previous analysis can be used to investigate the intersymbol dependence between the output of the (i-th) round and the 56 bit cryptographic key. However, since the keys used in the different rounds vary, the analysis is slightly more complicated. Since the condition $L(j+1) = R(j)$ introduces a certain symmetry, it is advantageous to evaluate the dependence of $L(j)$ and $R(j)$ on the key in addition to the dependence of $X(j)$ on the key. The details of the analysis are shown in Reference 5 and the results are quoted in Table III.

It is concluded that the minimum number of rounds is equal to five to achieve intersymbol dependence between ciphertext and key.

SUMMARY AND CONCLUSIONS

On July 15, 1977 an encryption algorithm, called the Data Encryption Standard (DES), became effective as a U.S. Federal Standard. The algorithm utilizes a secret cipher key (a sequence of bits) to transform 64 bits of plaintext into 64 bits of ciphertext.

One property of the DES is that each bit of ciphertext is a complicated function of all plaintext bits and all cipher key bits. This property, defined as "intersymbol dependence" was analyzed by evaluating how fast intersymbol dependence was achieved as a function of repeated mathematical operations which were defined as rounds. Each of these operations consists basically of substitution and transposition.

TABLE III.—Ciphertext/Key Intersymbol Dependence

Round j	Output/Input Relation		
	$L(j)$ vs Key	$R(j)$ vs Key	$X(j)$ vs Key
1	0.00	10.71	5.36
2	10.71	79.02	44.86
3	79.02	96.43	87.72
4	96.43	100.00	98.22
5	100.00	100.00	100.00

NOTE: Table entries express the degree of intersymbol dependence, i.e., the percentage of non-blank elements in the appropriate relations.

Three different forms of dependencies were considered. If an input bit affects the selection of a substitution function in one round, the corresponding output was said to have an autoclave dependence on the input.

If an input bit affects the argument of a substitution function in one round, the corresponding output was said to have a message dependence on the input.

Since the basic operation involving substitution and transposition is repeated several times, the functional relation between a ciphertext bit and plaintext bit can, in addition to the already defined relation, be a combination of both.

It was shown that after five rounds each ciphertext bit depends on all plaintext bits via message as well as autoclave dependence. In addition, a similar analysis revealed that each ciphertext bit depends on all key bits after five rounds.

The method applied to the DES is, in general, applicable to ciphers which (1) split up the input into two parts, (2) operate on one part first, and (3) combine the results of that operation with the other part using modulo 2 addition.

REFERENCES

1. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, 28 1949, pp. 656-715.
2. Feistel, H., "Cryptography and Computer Privacy," *Scientific American*, Vol. 228, No. 5, May 1973.
3. *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, 1977.
4. Ehrsam, W. F., S. M. Matyas, C. H. Meyer and W. L. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," *IBM Systems Journal*, Volume 17, No. 2, 1978.
5. Matyas, S. M. and C. H. Meyer, *Computer Data Security Through Cryptography* (to be published).

Data dependent keys for a selective encryption terminal

by ROBERT FLYNN
Polytechnic Institute of New York
Brooklyn, New York

and

ANTHONY S. CAMPASANO
AIL, A Division of Cutler-Hammer
Deer Park, New York

INTRODUCTION

The concept of a data dependent key for use in conjunction with a selective encryption device is presented. The use of an encryption key extracted from the data or a function of extracted data eliminates the problems associated with a statistical assault on encrypted data which was all encrypted using the same key. If this key extraction is done in hardware then the security of the key is reduced to the physical security of the device which encrypts/deencrypts and not on the personnel doing the data entry. This work is premised on and builds from work done by Carson, Flury, Summers and Welsh,[1,2] establishing the concept of selective encryption terminals as a feasible technology.

With the passage of the Privacy Act of 1974,[3] a new area of concern has been presented to the data processing industry, i.e., the required protection of data on individuals.

The recently published report on the Privacy Protection Study Commission[5] goes further. ". . . the ability to search through . . . records to identify individuals with particular characteristics of interest is at once the most important gain, and the most important source of potential harm, stemming from the automation of large scale personal-data-record-keeping systems, and its importance is likely to grow in importance as 'textual search' techniques are refined."[6]

In light of this, various methods of protecting data have been suggested. Among these is the adoption by the National Bureau of Standards of an "Encryption Algorithm for Computer Data Protection." The algorithm is designed to encipher and decipher blocks of data consisting of sixty-four bits under control of a sixty-four bit key.[4] This algorithm was implemented in an experiment performed by the METREK Division of the MITRE Corporation in 1975 and 1976. They developed the Selective Encryption Terminal which consists of a Tektronix 4023 terminal and an LSI-11 microcomputer in communication with a host computer where the database was stored. For this experiment the host was an IBM 370/145 running the VM/CMS operating system.[1,2]

SELECTIVE ENCRYPTION

The concept of selective encryption allows the terminal operator to encrypt only those portions of an individual's record for which protection is required or to encrypt the identification portion of a record so that the individual is not associated with the data. The latter is useful in databases where statistical studies are made with respect to certain aspects of the individual's files. Selective encryption also allows the terminal operator to change the key so that different sections of an individual's record can be encrypted with different keys. In the METREK experiment, a hospital database was used. Here, medical information was encrypted under one key for use by physicians while accounting information was encrypted under another key for use by the hospital's administration.

The most important feature of the Selective Encryption Terminal is best expressed in this excerpt from the METREK report.

> *The protected material remains in encrypted form throughout its residence in the host system and is decrypted only upon return to the terminal. In this manner, neither the protection (encryption/decryption) software, the protection key, nor the plaintext of the protected information is ever present in the host computer.*[1]

The plaintext referred to is the data before encryption.

VULNERABILITY OF THE KEY

It is important to note that the security of the METREK system and any other system utilizing the NBS algorithm is entirely dependent on the security of the key or keys. Given the information that the NBS algorithm was being used, an outside intruder could get access to the protected data by one of three ways. First, the key could be obtained by trial

and error: iterating through all possible values of the key until found. Second, the encrypted text could be analyzed statistically so that any patterns in the data might reveal the key. And finally and most simply, the key could be obtained from those personnel who are exposed to the key, such as terminal operators.

What is to be considered then, are variations in the implementation of the algorithm which would combat these problems. One such variation is the concept of using data dependent keys in performing the encryption. Discovering the key by trial and error even when the data dependent keys are not used is a difficult task. With an eight byte key, each byte an ASCII character in which only seven of the eight bits vary, there are fifty-six bits which numerate all possible keys, and if it were possible to process each attempt in one microsecond, it would take on the average over one thousand years to find the key.[7] The problem occurs if the intruder trying to break the system gets lucky or utilizes some fast and advanced multiprocessing system. With the key found, the entire database is at his disposal.

DATA DEPENDENT KEYS

Consider, rather, a data dependent key. For each record of data in the database, the key entered by the operator could be used to select certain bytes of the plaintext data which will not be protected by encryption in the record, which could then be used to derive the key for the encryption of the data to be protected. In this way, if the key used to encrypt one record was discovered, that specific key would be useless in decrypting any other records.

It is assumed, of course, that the data dependency of the key is kept as confidential as the key itself. However, if such is not the case, the data is still fairly well protected. Given that the eight bytes of the key are selected directly from the plaintext data with replacement, it would take on the average $(N^8)/2$ attempts to break the key where N is the total number of plaintext unprotected bytes available. If N were one hundred bytes and each attempt took one microsecond, the data would be protected for an average of over one hundred and fifty years. It should also be noted that the one microsecond figure is unreasonable for the technology available. The best figures to be found are those reflecting the implementation of the algorithm using T^2L devices. Here the time to encrypt sixty-four bits is five microseconds.[7]

Protection using the above scheme could be further improved by not using the plaintext data itself as the key but rather as input to a function which creates some function of a key from the data chosen. A simple arithmetic maneuver, such as a hashing code, could increase protection manyfold while not significantly affecting processing overhead. Different functions could be used for different fields and the types of functions used could be varied. Thus one function might have the property of creating widely scattered keys from varying plaintext. Such a key function would have a high velocity. Another function might be a highly noninvertible function (not one-to-one) such as adding the low order bits of the plaintext bytes.

Data dependent keys improve security against the second method mentioned, discovering the key by statistical methods. With each data record encrypted with a different key, the amount of encrypted data in each record would be too minimal to draw any conclusions about the key.

MASSIVE DATA ENTRY SYSTEMS—ENCRYPTING THE SOURCE

For a large database where data is entered by many different people but only retrieved by a few, another scheme might be in order. The premise here is that some data is available to all for statistical purposes but only a few could examine or identify a specific individual. Most massive data entry systems employ the key to disc concept. Many operators enter records at many terminals which tie into one processor storing the data on a disc. If this processor is physically local to the terminals, i.e., there is no security problem due to lines which can be tapped, then the encryption can be done at the processor and not at the terminal.

Given this configuration, the data can be encrypted at the key to disc processor using some subset of the identifying fields of the record, such as the name and/or the social security number, as the encryption key. With this key the protected records including the identifying fields can be encrypted. To retrieve the data, a special decryption terminal can be designed which, when given the identifying information used to encrypt the record, can determine the key in the same manner and retrieve the record. Here one would extract the plain text seed for various key functions. Encrypt the fields that are to be protected and then encrypt the plaintext header information (name, social security) using as a key a function of extracted subfields of the plaintext header data.

In this manner, only an individual authorized to use a decryption terminal could enter the specific name and social security number and from this extract the records on that individual. No one else could ever identify uniquely those individuals who had characteristics of interest. The data base is essentially not invertible and yet certain fields which are not encrypted could be made available to everyone for analysis.

This scheme is superior to those previously discussed in that there is no plaintext stored in the database. Furthermore, each encrypted record is encrypted with its own key, making attempts to intrude on the database via statistical methods futile. And perhaps most important, the data entry personnel do not see the encryption and may not be aware that it exists and do not have any retrieval capability at their terminal at all.

HARDWARE IMPLEMENTED KEYS

Protecting the key from the third source of disclosure, disloyal personnel, is probably the more difficult problem to solve. Personnel, such as terminal operators who are readily exposed to keys in use as well as some information concerning the implementation of the encryption are the weak-

est link in the security system. Depending on the terminal used, information concerning whether the key entered at the terminal is used to derive a data dependent key or applied directly as the encryption key might easily be obtainable through the keyboard. What is needed here then is a scheme by which the operator may utilize the encryption/decryption facilities of the terminal but without ever allowing disclosure of the key or the details of the implementation.

One such scheme would involve entering the key at the terminal using hardware. It is possible to fashion an intelligent terminal with insertible ROM's (Read Only Memories) similar to those used on Hewlett Packard's 9820 calculators. HP 9820 users can purchase various pre-programmed ROM's which when inserted in the calculator give the user access to a library of subroutines.[8] A terminal with a microprocessor could be so equipped and utilize subroutines on the ROM for encrypting, decrypting, deriving keys from data and supplying the main key itself to these routines on the ROM. What is important to note about the architecture of the HP 9820 is that the information on the ROM can only be used as instructions for the processor. The code in the ROM is not available to the user as data. Data and user supplied programs can only be entered and retrieved from the calculator's RAM (Random Access Memory). It is in the RAM where the program which calls the ROM subroutines resides.

Given a terminal with the mentioned specifications, the key and the method of implementing the encryption can now be physically protected. The ROM could easily be designed so that it can be locked into the terminal for safety only to be removed by some authorized personnel. All that the operator will see is the data being processed at the terminal.

It might be useful to consider designing the terminal with two ROM ports for use when converting an entire database from encrypted under one key to encrypted under a new key. A ROM in one port would be used to re-encrypt the data.

Security, of course, would be required when the ROM program is burned in. Once the implementation of the algorithm is established, a number of ROMs could be prepared at one time leaving the key field blank to be entered when needed and at a proper time and place. There should also be some method of sealing the ROM.

Another variation of this scheme would make use of recently developed semiconductor devices, such as the Motorola "Info-guard,"[9] to perform the NBS encryption/decryption of the data. If this device were already present in the terminal and its capability available to the data processing program in the RAM, the ROM need only contain the routines for deriving and/or supplying the key to the devices. Using a dedicated device rather than a general purpose processor to perform encryption/decryption generally implies that one could expect much faster processing time.

CONCLUSION

The Selective Encryption Terminal is by far one of the most promising developments for Privacy Protection. The data is secure from the moment it leaves the terminal. By simply supplying extra protection to prevent disclosure of the encryption key using either data dependent keys and/or a hardware implemented key, a data base is well protected from any accidental disclosure or unfriendly intrusion. The protection of data is now reduced to the physical protection of a device.

Using data dependent keys for encryption and supplying a key to the terminal via ROM instead of through the keyboard may seem awkward in terms of processing overhead and generating the ROMs themselves. However, each contributes significantly in spoiling a possible intruder's plans to a point where any attempt would be completely discouraged. These ideas, then, appear to be well worth developing. They may provide an attractive solution to the simultaneous needs of privacy and statistical reporting in government applications.

REFERENCES

1. Carson, J. H., W. R. Flury and J. S. Welch, "The Selective Encryption Terminal: A New Approach to Privacy Protection," M75-76, The MITRE Corporation, METREK Division, September 1976.
2. Carson, J. H., J. K. Summers, J. S. Welch, Jr., "A Microprocessor Selective Encryption Terminal for Privacy Protection," *Proceedings of the National Computer Conference,* AFIPS Press, Montvale, NJ, 1977, pp. 35-38.
3. "The Privacy Act of 1974," Public Law 93-579.
4. "Encryption Algorithm for Computer Data Protection," National Bureau of Standards, *Federal Register,* Vol. 40, No. 52-Monday, March 17, 1975.
5. Personal Privacy in an Information Society, the report of the Privacy Protection Study Commission, U.S. Printing Office, 1977.
6. Technology and Privacy, Appendix 5 to Personal Privacy in an Information Society (separately published), U.S. Printing Office, 1977.
7. Sykes, D. J., "Protecting Data by Encryption," *Datamation,* August 1976, Vol. 22, No. 8, pp. 81-85.
8. "Hewlett-Packard 9820A Calculator Operating and Programming Manual," Hewlett-Packard Company, Calculator Products Division, 1972.
9. "Computer Threats," Motorola Inc., Government Electronics Division, 1977.
10. "NBS Standard Puts Encryption in DP System," *Computerworld,* June 7, 1976. (not cited)
11. Hoffman, L. J., *Modern Methods for Computer Security and Privacy,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977. (not cited)

Security in communication networks

by MARTIN E. HELLMAN
Stanford University
Stanford, California

INTRODUCTION

It may seem anomolous that electronic mail and other computer communication systems require cryptographic protection when almost no physical mail is given this protection. The difference is that computer readable traffic is extremely vulnerable to automatic sorting at very low cost. Physical mail would also need to be encrypted if it were all written on postcards and could be sorted at a cost of only $1 for several million pages. Even seemingly innocuous facts can be damaging when such vast amounts of data can be screened for all messages mentioning one of a list of key words (e.g., computer communications, electronic mail, EFT, etc.). Analog voice circuits are as vulnerable to wiretapping, but are expensive to sort.

Fortunately, the digital nature of the data makes high grade encryption possible at low cost. Analog circuits are almost impossible to adequately secure without going through a digital interface and encryption. The National Bureau of Standards has promulgated a national data encryption standard which can be implemented on a single LSI chip.[1] In large quantities it should therefore cost on the order of $10, an insignificant addition to the cost of a computer terminal. While some have criticized the standard as being inadequately secure,[2-4] this is not due to technical constraints, but rather appears to be a political problem.

While the cost of the encryption hardware is not a barrier to the widespread use of cryptography in computer oriented systems, there are other costs and problems which must be considered. Key distribution is one such problem.[5] In a network with n users there are approximately $n^2/2$ possible pairs of users who may wish to converse securely from all other users. The distribution of this many keys by courier, registered mail, etc. is clearly uneconomic even for n equal to one million. This problem can be solved by having the system itself distribute keys, encrypted in user specific system keys or passwords, but this requires the system to be secure.[6,7] A more useful approach was suggested by Diffie and Hellman[5] and Merkle.[8] They proposed that it is possible to converse securely over an insecure channel with no prearrangement through use of "public key systems." The second section describes the public key systems of References 5 and 8 as well as systems devised by Rivest, Shamir and Adleman,[9] McEliece,[10] and Merkle and Hellman.[11]

While none of these public key systems has been broken thus far, it is necessary that they withstand the test of time and concerted mock attacks by dedicated "opponents" before they can be trusted because no methods are currently known for proving even conventional cryptosystems secure. In this regard we applaud the work of Simmons and Norris[12] which looked for potential weaknesses in one of the public key systems. More such work is needed, and it would not be surprising if weaknesses were found in one or more of the currently known systems, much as conventional cryptography also went through a learning period.

Digital signatures are discussed in the third section. Conventional cryptographic systems can prevent third party forgeries, but cannot settle disputes between the transmitter and receiver (e.g., a stock broker and his client) as to what message, if any, was sent. Solutions to this problem were first hypothesized by Diffie and Hellman[5] and found by Rivest, Shamir and Adleman[9] and Merkle and Hellman.[11]

PUBLIC KEY SYSTEMS

Merkle's public key system[8] is perhaps the simplest and least likely to yield to continued cryptanalytic efforts. Its disadvantage is that it is the most expensive. It depends on the existence of a one way function, a function that is easy to compute for all arguments in its domain but computationally infeasible to invert for almost all images in its range. Such functions have been discussed elsewhere[13-15] and are as easy to develop as secure cryptosystems of the conventional, as opposed to public key, type.[5] Merkle goes a step further and describes a method for generating one way functions of controllable "one wayness" or difficulty of inversion.

A mildly one way function can be used to generate a puzzle, a problem which is difficult, but not impossible, to solve and for which it is easy to check a supposed solution. User A generates m keys, $k_1, k_2, \ldots, k_m$, and operates on them with a mildly one way function $f(*)$ to obtain their images or puzzles, $y_1, y_2, \ldots, y_m$. These images are transmitted to user B who selects one of them at random, say y_i, and solves the associated puzzle to obtain k_i. The two users now share a key in common but A does not know which one it is. B therefore sends A the value $z=g(k_i)$, where $g(*)$ is a

true one way function. A then operates on k_1, k_2, etc. with $g(*)$ until he finds one which yields z. The successful key must be k_i provided $g(*)$ is one-to-one.

The cost to A is linearly proportional to m, the number of puzzles, because A must operate on m k's with $f(*)$—which is easy—to obtain the m y's; he must store the m k's; and finally must operate on approximately $m/2$ of the k's with $g(*)$—again an easy task—before finding z as an image. A's transmission cost is also linearly proportional to m since he must send m y's. B's dominant cost is in solving the one puzzle that he chose at random. By making the cost of solving a single puzzle proportional to m, the total cost to the two legitimate users is still only linear in m.

An eavesdropper, however, must solve $m/2$ puzzles on the average before finding k_i, so his cost is proportional to m^2. Thus as m tends to infinity the ratio of cryptanalytic cost to key exchange cost also tends to infinity.

This method's weakness is in the relatively small ratio of costs (m^2:m) and the fact that the key exchange cost is as much in transmission as in computation. (Transmission costs have not decreased as rapidly as computation costs.) The introduction of low cost, high bandwidth transmission media, such as fiber optics, may make this method more competitive. Merkle's paper[8] describes several very clever additions to this simple description, but the basic cost ratio is not changed.

Diffie and Hellman[5] propose a method of public key exchange which requires $2^{b/2}$ operations for cryptanalysis (using the best known algorithm) but only b^3 operations for key exchange, where b is the number of bits in the representation of the key. By choosing $b=400$, key exchange requires only 64 million gate operations and takes approximately one second in a special purpose LSI implementation. Cryptanalysis using currently known techniques requires approximately 10^{60} operations and words of memory and is therefore totally infeasible. This technique makes use of the apparent one wayness of the discrete exponential function $y=a^x \bmod q$, where q is a large prime number of appropriate form [16,17] and a is a fixed primitive element of the finite field $GF(q)$. Calculating y from x—with tacit knowledge of a and q—is relatively easy, and requires only three words of memory, each b bits long, and b^3 gate delays. Computing x from y is believed to be much harder, and the best known algorithm[16,17] requires memory and time proportional to $2^{b/2}$.

The two users and the cryptanalyst are assumed to know q and a. Each user generates a random number uniformly distributed between 2 and $q-2$. Call these values x_1 and x_2. The users keep these values secret, but compute $y_1=a^{x1} \bmod q$ and $y_2=a^{x2} \bmod q$ and exchange these values. The cryptanalyst therefore also learns y_1 and y_2, but cannot feasibly compute x_1 or x_2 therefrom. User 1 takes y_2 (which was sent to him) and x_1 (which he has kept secret) and computes $(y_2)^{x_1}=(a^{x_2})^{x_1}=a^{(x_1x_2)} \bmod q$. User 2 computes $(y_1)^{x_2}=(a^{x_1})^{x_2}=a^{(x_1x_2)}$. Both users are now in possession of a common number $K=a^{(x1x2)} \bmod q$ which they use as the key in a normal cryptographic system. The cryptanalyst cannot compute K as any obvious function of y_1 and y_2 (e.g., y_1y_2 or $(y_1)^y2$) without first computing either x_1 or x_2, which is an infeasible task using the best known algorithms. There may be better algorithms for computing x_1 and x_2, or there may be some nonobvious method for computing K from y_1 and y_2 directly. As with all cryptographic systems, this one should be studied further to increase our trust in it.

Merkle and Hellman[11] have proposed a public key method based on trap door knapsacks. Given a one-dimensional knapsack of length S and a set of n rods of lengths a_1, a_2, . . . , a_n, one version of the knapsack problem is to find a subset of the rods whose lengths sum to exactly S. Equivalently, find a binary n-vector x such that $\mathbf{a}^*\mathbf{x}=S$. (The dot product of two vectors is denoted by *.)

The knapsack problem is believed to be very difficult in general, and this belief is supported by its being an NP-complete problem.[18] In a loose sense the NP-complete problems are the most difficult problems of a cryptographic nature.[5] A trap door knapsack vector $\mathbf{a}$ is one which has no apparent structure which can be used to simplify the solution process, but which possesses hidden (trap door) structure which allows rapid solution for $\mathbf{x}$.

As a small demonstration example, consider $\mathbf{a}$=(5457, 1663, 216, 6013, 7439) and $S=1663+6013+7439=15115$, corresponding to $\mathbf{x}$=(0,1,0,1,1). It happens that if each component of $\mathbf{a}$ is multiplied by 3950 mod 8443 (the secret, trap door information) the vector $\mathbf{a}'$=(171, 196, 457, 1191, 2410) results. This vector has the property that each component is larger than the sum of all the preceding components. Transforming S in a similar manner (multiplying it by 3950 mod 8443) yields $S'=3797$. Some thought shows that the solution to the problem $S=\mathbf{a}^*\mathbf{x}$ is the same as the solution to $S'=\mathbf{a}'^*\mathbf{x}$, and that the solution to $S'=\mathbf{a}'^*\mathbf{x}$ is easily found because of the form of $\mathbf{a}'$. x_5 must be 1 because $S'\geq a_5'$—if x_5 were 0 then even if all other components of $\mathbf{x}$ were 1's the sum could not be large enough to yield S'. Subtracting a_5' from S' yields $3797-2410=1387$ which is the sum of a subset of the remaining components of $\mathbf{a}'$. Because $1387\geq a_4'=1191$ we know that x_4 must also equal 1. Subtracting a_4' from 1387 yields 196. This is smaller than $a_3'=457$ so x_3 must equal 0. It is equal to $a_2'=196$ so x_2 must equal 1 and x_1 must equal 0. The determination of $\mathbf{x}$ is now complete and, as a quick check will show, correct for the original problem $S=\mathbf{a}^*\mathbf{x}$ as well.

Of course the trap door knapsack vector $\mathbf{a}$ was not generated first. Rather $\mathbf{a}'$ was first chosen with the property that each component was larger than the sum of all preceding components and then transformed into the $\mathbf{a}$ vector by multiplying each component of $\mathbf{a}'$ by 2550 mod 8443 (2550 and 3950 are multiplicative inverses mod 8443). In a similar manner a program could easily be written to generate rather large trap door knapsack vectors from a random bit string. Any user of a computer system could then generate his own personal trap door knapsack vector regardless of his mathematical abilities. The program would also generate the secret multiplier and modulus which reduces the apparently difficult knapsack problem $S=\mathbf{a}^*\mathbf{x}$ to the trivial problem $S'=\mathbf{a}'^*\mathbf{x}$. This program is assumed to be public knowledge but, even so, there is no apparent way for a cryptanalyst to easily solve for $\mathbf{x}$ only from knowledge of S and the public vector $\mathbf{a}$.

After generating a trap door knapsack vector $\mathbf{a}$, the user

can place it in a public file. Then anyone who wishes to send him information can do so by representing it in binary blocks of n bits each, and using these as **x** vectors to compute the sums $S=\mathbf{a}*\mathbf{x}$ which are sent to the first user, who can easily recover the information **x** even though no one else can.

Note that this system is different from either of the first two public key systems in that a normal cryptographic system is not needed. This is because the first two systems each generated a number that the two legitimate parties to the conversation could easily compute, but neither of the parties could determine that number on his own. In the trap door knapsack system **x** is determined entirely by one of the users. While it is not necessary for **x** to be used as the key in a normal cryptographic system, in practice, the speed advantages of conventional cryptographic systems will probably cause **x** to be used in that manner. This same remark applies to all of the currently known public key systems.

The public key system due to Rivest, Shamir, and Adleman[9] can be regarded as a generalization of a conventional cryptographic system developed by Pohlig and Hellman.[16] Each user generates a pair of numbers E and n which are placed in a public file and which are used by others to encipher data they wish to send him. At the same time that E and n are generated, another number D is generated which is required for deciphering data. Clearly, it must be computationally infeasible to compute D from the public information E and n if the system is to be secure. As shown in Reference 9, computing D from E and n is equivalent to factoring n, and it is possible to choose n so that this is infeasible using the best known factoring algorithms.

First two large prime numbers p and q are chosen and multiplied to produce $n=pq$. Then Euler's function $m=\text{phi}(n)=(p-1)(q-1)$ is computed. Phi(n) is the number of integers between 1 and n which are relatively prime to n, and has the interesting property that almost any number between 1 and n when raised to the m power mod n equals 1 (the exceptions turn out not to affect the system and we therefore neglect them in what follows.[9]). E is then chosen as a random number between 1 and m which is relatively prime to m, and D is computed using Euclid's algorithm to be the multiplicative inverse of E mod m. That is $ED=km+1$ for some integer k. Enciphering requires only one exponentiation in modulo n arithmetic and is easily accomplished. Letting P denote the plaintext and C the ciphertext, $C=P^E \bmod n$. (The plaintext must be represented as a sequence of integers each between 0 and $n-1$.) Deciphering is also easily accomplished in one exponentiation, $P=C^D \bmod n$. To see that this really does undo the enciphering operation note that $C^D=(P^E)^D=P^{ED}=(P^m)^k P^1=P \bmod n$, because $P^m=1$.

The most recently developed public key system is due to McEliece,[10] and is based on algebraic coding theory. Goppa codes are highly efficient error correcting codes, both in their error correcting capacity and in the computation required to correct errors. The ease of error correction is destroyed, however, if the bits which make up a codeword are permuted prior to transmission. In McEliece's system a user's public enciphering key describes a scrambled Goppa code, chosen at random from a large set of possible codes. Anyone can easily encode information (scrambling the Goppa code does not greatly affect the ease of encoding because the code is still linear), add a randomly generated error vector and transmit this to the user. But only the intended recipient knows the inverse permutation which allows the errors to be corrected easily. McEliece estimates that a block length of 1000 bits, with 500 information bits, should foil cryptanalysis using the best currently known attacks. The main problem is the storage of a 500 by 1000 bit generator matrix, requiring 500 kilobits of memory per user.

SIGNATURES

Written signatures are essential to our current methods of conducting business. They serve to indicate accountability and agreement on contracts, etc. Before electronic means can fully replace physical (hardcopy) forms of information, a digital equivalent to a written signature is needed. A digital authenticator must be a number which is easily recognized without being known, because any number that is known can be forged by the intended recipient. While at first appearing to be a logical impossibility, digital signatures can be obtained from public key cryptosystems and probably in many other ways as well.[5]

Rivest, Shamir, and Adleman's system[9] yields signatures most directly, merely by interchanging the enciphering and deciphering operations, so we only describe that method. When a user wishes to send a signed message M to someone else, he operates on it with his secret key D to obtain $Y=M^D \bmod n$. The recipient can recover M through use of the public key E,n because $Y^E \bmod n=M$. The recipient saves Y as proof that message M was sent to him by the user whose public key is E,n. If the sender later disclaims having sent the message, the recipient gives Y to a "judge" who can access the public file and see that $Y^E \bmod n$ does in fact equal a meaningful message with the right header information. Only the user who placed E,n in the public file knows D and could produce such a Y.

In practice each block of the message will probably not be signed in this manner. Rather, to speed things up, the message will be sent in its untransformed state, and a one way hash total H of the message computed.[5] The signature will be $Y=H^D \bmod n$. The recipient can easily check that H results when the public key E,n acts on Y, and that it is the same H as obtained from action of the hash function H on the message.

The above discussion neglects the privacy problem which results if an eavesdropper may be listening. This problem is easily overcome by enciphering the message-signature combination in a normal or public key system.

CONCLUSIONS

Public key systems and digital signatures make teleprocessing systems vastly more useful for business and personal

use, but care must be exercised, both at the technical and legal levels, to ensure that these advances are not used in a detrimental manner. For example, a user's secret key will probably be stored on a magnetic card which is needed to transact any business on the system. If the system becomes all pervasive in daily life, people may be expected to carry their cards with them constantly. It is only a small step to allow the police to demand the card as a form of universal identifier, without which a person becomes a nonperson. There are clearly dangers in such a system and adequate safeguards must be built in. Even now, certain businesses (e.g., car rental, gas stations at night) will accept only credit cards.

Further research is obviously needed at a technical level. The security levels of the currently known systems need better evaluation and new systems should be sought. These may be more efficient than the currently known systems, or needed in the unlikely event that holes are found in all of them. A major research goal is the establishment of provably secure systems, conventional, public key, and signature. That goal is more ambitious than solving one of the premier outstanding problems in computer science (the P=? NP problem) and must be viewed as long term.

REFERENCES

1. National Bureau of Standards, *Data Encryption Standard,* Federal Information Processing Standards Publication 46, January 1977.
2. Diffie, W. and M. E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer,* June 1977, pp. 74-84.
3. G. B. Kolata, "Computer Encryption and the National Security Agency Connection," *Science,* Vol. 197, July 29, 1977, pp. 438-440.
4. Morris, R., N. J. A. Sloane and A. D. Wyner, "Assessment of the National Bureau of Standards Proposed Federal Data Encryption Standard," Bell Laboratories Memorandum, November 1976. Also in *Cryptologia,* Vol. 1, July 1977, pp. 281-291.
5. Diffie W. and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. on Info. Theory,* Vol. IT-22, November 1976, pp. 644-654.
6. Brandstad, D., "Security Aspects of Computer Networks," *AIAA Computer Network Conference,* April 1973.
7. Diffie, W. and M. E. Hellman, "Multiuser Cryptographic Techniques," *National Computer Conference,* AFIPS Conference Proceedings Vol. 45, June 1976, pp. 109-112.
8. Merkle, R. C. "Secure Communication Over an Insecure Channel," to appear *CACM*, April 1978.
9. Rivest, R., A. Shamir, and L. Adleman, "On Digital Signatures and Public-Key Cryptosystems," to appear *CACM,* February 1978.
10. McEliece, R. J. "A Public Key System Based on Algebraic Coding Theory," *JPL DSN Progress Report,* 1978.
11. Merkle, R. C. and M. E. Hellman, "Hiding Information and Signatures in Trap Door Knapsacks," to appear *IEEE Trans. on Info. Theory.*
12. Simmons, G. J. and M. J. Norris, "Preliminary Comments on the MIT Public-Key Cryptosystem," *Cryptologia,* Vol. 1, October 1977, pp. 406-414.
13. Wilkes, M. V. *Time Sharing Computer Systems,* Elsevier, New York, 1972.
14. Purdy, G. B. "A High Security Log-In Procedure," *CACM,* Vol. 17, August 1974, pp. 442-445.
15. Evans, A. Jr., W. Kantrowitz, and E. Weiss, "A User Authentication System Not Requiring Secrecy in the Computer," *CACM,* Vol. 17, August 1974, pp. 437-442.
16. Pohlig, S. C. and M. E. Hellman, "An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance," to appear *IEEE Trans. on Info. Theory,* Vol. IT-24, January 1978.
17. Pohlig, S. C. *Algebraic and Combinatoric Aspects of Cryptography,* Ph.D. thesis, Stanford University, EE Dept., November 1977.
18. Karp, R. M. "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations,* R. E. Miller and J. W. Thatcher, Eds. Plenum, New York, 1972, pp. 85-104.

PART IV—PEOPLE AND SOCIETY

Area Director:
Susan H. Nycum
Chickering & Gregory
San Francisco, California

Legislation

OVERVIEW

Legislative and regulatory activity affecting the computer industry is on the increase. A recent computer search of legislation pending before the Congress retrieved 328 bills involving computers and/or communications. These bills and others pending before the fifty state legislatures supplement the numbers of existing laws that already affect the industry. Recent legislative and related regulatory activity has been in five general areas: communications including transborder data flow, privacy, protection of proprietary rights in software, personnel and security. (One specific computer application, electronic funds transfer, which is the subject of a related area, includes aspects of each of the above.) Lawyers and computer specialists will describe and analyze this activity and its impact on computing in the following sessions:

The session on communications entitled "Federal Policy And The Future Of Computer Communications Services" will focus on the present status of federal policy, regulation and legislation impinging on computer communications, as well as relevant anticipated developments and their impact on the data processing field. Discussion will include activity in the executive branch concerning international communications and the office of the Assistant Secretary of Commerce for Communications and Information. Considerations of the Federal Communications Commission in the ongoing FCC "Computer Inquiry" (Docket No. 20828) will be discussed and Congressional issues, particularly the possible revision of the Communications Act of 1934 and related legislation will be reviewed.

In a session entitled "Privacy: Practical Implications For Your Organization," a leader in the Federal Executive Branch Privacy Task Force will report on questions raised by that group's recent review of the privacy issues, an analysis of the technological implications of privacy laws, regulations and recommendations for organizations and projections of future development. Panelists will discuss the report from the viewpoints of technologists, individuals and business organizations.

The legal aspects of software protection will be discussed in a panel session in which the current status of patent, tradesecret and copyright law will be described by experts in the area—one of whom is also a Commissioner of the Federal Commission On New Technological Uses Of Copyrighted Works (CONTU). An industry executive will then comment on the kind of protection the software industry needs and inquire of the other panelists how current or proposed law responds to these needs.

Personnel issues in computing involve a variety of concerns.

Computer Crime: Sources, case studies and prevention will be the topic of the session in which three papers will be presented that describe and defend teaching EDP to prison inmates, detail a recent computer crime prosecution and suggest a new prevention strategy based on the differences between accidental and intentionally caused losses.

Software house in the big house

by WILLARD V. HANDLEY
Federal Prison Industries, Inc.
Winchester, Virginia

INTRODUCTION

At the United States Penitentiary, Leavenworth, Kansas, and the Federal Correctional Institution, Milan, Michigan, some fifty convicted felons are learning to be computer programmers. Inmates supplement academic training with experience by developing systems for the Veterans Administration, Department of Labor, and other Federal agencies. Programming services are sold to agencies at market prices to generate revenue for inmate salaries and other expenses. This custom software enterprise is, or course, managed and supervised by civil service staff.

A software house whose programmer staff are convicted felons must pay attention to the potential of computer fraud in its security program. But it is not enough to have a program deemed adequate by experts in the field. To the general public, the computer is still an awesome and mysterious device. Our operation, and indeed any computer operation susceptible to public opinion must, like Caesar's wife, not only be above reproach but appear above reproach.

We learned this fundamental lesson when an allegation was made that inmates were using our computers to defraud IRS. Nothing of the sort had happened, nor could happen, under the stringent procedures in force at Leavenworth. We knew it; but we could not convince our critics, because the explanation was too complex.

This paper explains how we subsequently changed our security program to make it more clearly obvious to any observer that inmates could not misuse our resources. It should prove useful not only to other prison systems with similar rehabilitation programs, but to any organization affected by public opinion. Everyone may not have inmates as programmers, but most have some form of Achilles' heel. The need to go beyond "secure" to "obviously secure" in your own sensitive area of security may be as great as was ours.

The IRS episode also resulted in criticism of the fundamental purpose of our program. There are some who argue that a skill so easily turned against society should not be taught to known felons. Although an inmate, once released from custody, is outside the scope of our security program, this paper would not be complete without an explanation of why we feel the program is a boon, rather than a bane, to society.

COMPUTER PROGRAMMING IN FPI

The Federal Prison System houses some 30,000 offenders in prisons located throughout the country. The Prison System has a duty beyond merely keeping *them* isolated from *us*. It is expected to afford inmates humane treatment and the opportunity for rehabilitation.

Humane treatment includes helping the inmate pass the endless hours without losing his sanity. This entails the need for meaningful work programs; ideally, these programs contribute also to rehabilitation by teaching a skill usable on the "outside."

"Rehabilitating offenders" is a phrase not now heard in the field of corrections as often as before. Now we admit the best we can do is provide opportunities for self-improvement. So, education, vocational training, religious opportunities, and work programs are made available, on a voluntary basis, along with close counseling by social workers. Some inmates sincerely avail themselves of these opportunities; others merely "do their time," and often it is these who come back to prison again and again.

The Leavenworth programming enterprise is only one of many self-supporting work/training programs operated by Federal Prison Industries, Incorporated. This wholly-owned government corporation was established by Congress in 1934 to operate as a private manufacturing concern, buying raw materials, producing products, and selling them to generate its operating revenue. Its management and supervisors are government civil servants; its workers, inmates. It was given freedom to develop and manufacture any product, but restricted to selling to other Federal agencies. It was capitalized with $4 million of machinery, equipment, and other assets with no provision for subsequent Federal funding.

Today, FPI, now operating under the trade name of "UNICOR," has seventy factories and shops in thirty-two penal institutions throughout the country. It is virtually a conglomerate, with six product line divisions and forty industrial activities. UNICOR makes textiles; several lines of office furniture; footwear; wiring harnesses for missiles, tanks and aircraft; operates print plants, sign factories, microfilming shops engineering drafting shops, and other graphic arts facilities; and offers data entry, systems analysis, and computer programming, and other commodities and services. It employs 5,900 inmates, 600 civilians, and has annual sales

of over $90,000,000. It has been self-sustaining for the forty-three years of its existence. Unlike some government corporations, and not a few private ones, it has never needed another infusion of Federal funds.

Probably the most ambitious UNICOR effort to provide inmates usable skills was the establishment, in 1972, of the Leavenworth computer programming industry. As with most UNICOR industries, its workers have little or no prior experience. A year of classroom training, taught by certified teachers, is provided to prepare carefully selected inmates for the program. Graduates then work in a closely supervised environment to increase their skills with live work.

Students are selected from among a usually long list of volunteer applicants who have been recommended by their counselors. The potential trainee must have a high school diploma or GED certificate, perform at the tenth grade level in general math and language arts, and score a minimum of 45 out of a possible 90 points on the IBM Programmer Aptitude Test. A panel of custody, counselor, and industry personnel reviews such documents as the inmate's prior criminal record, pre-sentencing report, caseworker's progress reports, and overall incarceration history to reach a decision on each applicant's suitability for the program.

The course consists of thirteen four-week instructional modules designed to provide the equivalent of vocational school programmer training. It includes hands-on training with an NCR Century 200 computer with 64k bytes of main memory, dual disk storage units, a card reader, and line printer. The system software includes the operating system, assembler, and both FORTRAN and COBOL compilers. The course stresses COBOL, although an introduction to the assembler language and to FORTRAN are included. One module introduces IBM JCL and operating system utilities.*

A new course begins roughly every six months to start a new class of fifteen students on the road to becoming programmers. By the end of the year, dropouts and elimination by rigorous testing has dwindeled the group to about 12. This number is adequate to maintain production staffing at about thirty programmers and eight analysts.

Production programmers are grouped into six or seven man teams, carefully staffed to maintain a balance of experienced and novice programmers. A non-working civilian supervisor, qualified by years of experience, heads each team. The leader is responsible not only for the quality and quantity of each team's production, but for each member's continued development and growth.

Additional staff required to operate the facility includes two Leavenworth-based analysts and two Washington-based analysts/marketing representatives. The Leavenworth analysts are augmented by a team of four inmate analysts each. An administrative officer and a computer room operator round out the staff.

The training program currently costs over $150,000 annually; operating costs for the production element bring the total to nearly $550,000. But with the potential for over 60,000 programmer hours per year, the industry is still capable of operating "in the black" despite the unusual need to extensively train every programmer.

When the program began in 1972, a customer was found conveniently close to Leavenworth: The Agriculture and Stabilization and Conservation Service in Kansas City. The ASCS sent Leavenworth detailed programming specifications which were reviewed and clarified by Leavenworth analysts before assignment to the teams. Leavenworth was soon turning out programs whose reject rate of 8.7 percent was deemed by the customer "much better than average." Inmates were honing their skills on live production work, and were beginning to find jobs in data processing upon release. By June 1976, twenty-four of the forty-eight released had become programmers or analysts in private industry.

It soon became apparent, however, that having one customer was not enough. To provide a continuous flow of work and keep all programmers busy most of the time, we needed several customers to fill in the gaps. Turning to other agencies, however, meant computer compatibility problems, since the Federal government, by design, has many different computers. Because of our proximity to ASCS/Kansas City, we had been able to use the customer's machine remotely; now we were looking at the distant market in Washington, D.C., where such arrangements might be impractical.

The obvious solution was commercial time-sharing. For each job, we would find a vendor who offered the same machine as our potential customer's, and an existing network to give us local access.

We started with Infonet and its UNIVAC 1108s. Inmates quickly learned to use time-sharing and the new tools it afforded, such as interactive programming. Infonet had not been chosen casually; it boasted more Federal customers than any other network. By contracting with these agencies, we could develop systems on the same computer as our customers thus eliminating compatibility problems.

That on-line access to computers shared with Internal Revenue, the Veterans Administration and others also posed the hazard of subversion was not overlooked. The initial, and as it turned out, only, terminal was placed in a secure room. Civilians made all telephone contacts, and observed inmates continuously as they used the machine. Plans for additional terminals included a PBX to be modified by the telephone company so that outgoing calls could be made only at the switchboard, in response to verbal requests by civilian staff. The switchboard would be located in the administrative area of the prison, outside the main security gates. All telephone lines would be disconnected at the switchboard at 4:00 P.M.

But we had not gone far beyond the training stage with this new tool before an unrelated incident occurred in another part of the prison that embroiled us in a national controversy, and eventually eliminated time-sharing as a resource of this program.

A CRISIS AND ITS IMPACT

Among Leavenworth's population at that time was an inmate busily defrauding the Internal Revenue Service.

* A copy of the curriculum is provided at the end of this paper.

There was nothing sophisticated in his operation; he was taking advantage of the commonly known fact that IRS does not crosscheck, except on a limited scale, the amount a taxpayer's W-2 form shows was withheld with what his employer reports. To his phony tax returns showing money he had not earned, the inmate simply attached bogus W-2 forms showing taxes that had never been withheld. IRS processed the returns routinely, and sent a refund check to his wife on the outside.

The inmate was eventually caught and prosecuted. Although he was not connected with our program in any way, it was perhaps inevitable that it would be assumed our computer was being used in this scheme. Overnight, the story spread and grew by leaps and bounds. In major newspapers, wire service reports, and national television news broadcasts, it was implied, and sometimes stated outright, that inmates trained by the prison were using computers to crack the IRS security codes, file hundreds of bogus returns, and embezzle millions of dollars.

Computer security experts' statements, such as "No computer system exists that cannot be penetrated" made it difficult for the press to understand our assertions that our inmates could not penetrate Infonet. No one wanted to hear that, as a practical matter, penetrating a system requires certain resources, such as a prior knowledge of the operating system and unlimited access to a terminal, neither of which our staff had. In the end, we suffered as much adverse publicity as had we been guilty.

FPI Security

The Leavenworth security program had to counter three distinct possibilities for computer fraud at the time of the IRS scandal:

1. *Client System Penetration.* An inmate might penetrate the Infonet system of controls and gain unauthorized access to sensitive data of the FPI client or other Infonet users. He might also modify copies of programs stored on Infonet so that they would produce something of value to himself at some future time. Likewise, he might penetrate the ASCS IBM 360 machine to which the Leavenworth computer was linked.
2. *Client System Exploitation.* An inmate might produce a program that, in addition to its intended function, secretly provided some benefit, such as issuing a check to the programmer, or indirectly benefited him by alerting the client's operating system to subvert his security controls for later unauthorized use.
3. *Unauthorized Use of Computer Resources.* An inmate might use the Leavenworth computer or the Infonet system to directly produce something of value for himself, such as an official-appearing transcript; or to run a lottery.

These same threats exist in most data processing organizations.[1] We felt at the time that, while we should do everything possible to reduce the probability that these threats would be carried out, we would be doing nothing more than any conscientious software house should be doing.

Our countermeasures at that time were based on controls that eliminated the personal freedom and access to equipment needed to carry out the threats previously described. A large measure of control was provided as a byproduct of being in a prison setting, and of having a staff comprised of trainees and "green" programmers.

An important factor in all publicized computer crimes to date, and indeed, in virtually all white collar crime, is the numerous privileges and freedoms enjoyed by the perpetrators.[2,4] But the incarcerated Leavenworth inmate leads a life of almost constant supervision. He is deprived of the usual freedoms, not out of fear he will commit a computer crime, but because his custody is the responsibility of some prison employee at all times. He must have express permission to move from one area of the institution to the other, and must always be in a prescribed place at a prescribed time. At night, he is locked away in a cellblock separated from the computer room by numerous barred doors and gates.

During working hours, he has the freedom to be in the computer programming area, but still the requirements of custody apply. His supervisor must be constantly aware that he is present and doing his assigned job. No "outside" programmers are so closely observed.

Few would contest the statement that in the usual programming environment, managers are not enough in touch with their programmers to constantly know what they are doing. Thus a programmer runs little risk of discovery in appearing to be doing legitimate work while actually utilizing the computer for fraudulent purposes.

A close relationship between team leaders and programmer is assured at Leavenworth because the primary mission is different from that of the public sector software house. The supreme goal here is to build each inmate into an accomplished programmer so that he can succeed in the field upon release, rather than to make money for the enterprise. The team leader works closely with the inmate from the initial stages of specification interpretation, through flowcharting, debugging, and testing. This is a well-recognized concept for promoting program integrity, which in turn helps to ensure that a program does only what it is intended to do, and nothing more.[1,3]

Thus, the environment in which this software house operates provides a degree of control not usually found in the public sector. But additional, more direct security measures were also in force at the time of the IRS incident to counter the three main threats:

1. *Management Control of Computer Runs*—All run decks are submitted to the inmate's team leader, who reviews each deck to ensure that the job is one assigned to the inmate, and that the JCL meets prescribed standards: Nothing is suppressed on the output listings, no permanent files are created, and only the test files prepared by civilian staff are accessed. Decks may be brought into the locked computer room only by the team leader, who also must retrieve the printed output.

Every listing is reviewed by the team leader before being turned over to the programmer.

2. *Limiting the Possibilities*—Live test data is never introduced into the prison, nor were inmates given access to live data when shared computers were being used. An unwritten policy at the time of the incident, since formalized, was to avoid accepting systems to be programmed that provided an opportunity for financial gain.
3. *Personnel Selection*—The screening process previously described for inmates being considered for entry into the program was designed not only to ensure that the man had the ability for the program, but, to the extent possible, that he had a sincere desire to change his life for the better. Admittedly, this judgment is subjective.
4. *Independent Testing and Review of Programs*—ASCS policy required that all programs be compiled and retested with their own test data against the program specifications and flowcharts. ASCS also ran each program through the Performance Monitor of METACOBOL, a product that helps in determining program integrity by listing unexecuted paragraphs.

Most professionals would agree that the above procedures, if consistly and conscientiously applied, would come as close to eliminating the possibility of computer fraud by Leavenworth inmates as possible, given the existence of the threats outlined. Indeed, a detailed analysis of this environment by the MITRE Corporation, well-known for its efforts in security on behalf of the Department of Defense, led them to state in their first report following the IRS incident: "The security procedures in use at this time are probably stronger than those used in most software development shops, and are more than adequate for the types of programs currently being developed and the current development system."[4]

Nevertheless, we realized that this was not enough to prevent a repetition of the incident and its consequence of adverse publicity. An indirect threat existed for which we had no countermeasure: The public outcry against a danger more perceived than real. It could spell the end of our program. A combination of a general belief that with computers, anything is possible, and the prejudice that everyone who has previously committed a crime will do it again, given an opportunity, meant the issue of computer fraud was extremely sensitive in our situation.

The conclusion was that countermeasures against the three threats was not enough; we had to go further and eliminate as many of the threats themselves as possible. Under the guidance of the MITRE Corporation, we moved to reduce all threats to the issue of program integrity, and then to maximize our countermeasures against it.

This is being accomplished by isolating the development process from the client or target system. We withdrew from Infonet and will sever the connection with the ASCS machine, and begin using our own dedicated machine for system development. Thus, the only thing that will flow between the development facility and the client system is the program itself, which will be moved manually. The 'dedicated facility' is a well-known security technique, recognized by Department of Defense.[5]

The residual issue of 'program integrity' was addressed by strengthening the procedures already in effect. The technique of structured programming was adopted to help expose faulty or malicious code. A comprehensive, but common-sense, set of procedures and controls was also adopted to protect the program from the time it is "cleared" by the Leavenworth staff until installed on the target system. To further reduce the threat of program integrity, we have defined a set of formal guidelines against which to screen future client systems to reduce as much as possible the chance that an inmate will work on a program that could benefit him. These exclude: Programs that directly update data files that are used as the basis of financial transactions; programs in any way involved in the disbursement or transfer of funds to individuals, organizations, or accounts; and programs that access files which contain classified or extremely sensitive data that could be used for immediate or future financial gain.

All of these procedures are designed to protect society from the inmate programmer still incarcerated. They of course have no effect of the inmate programmer once he is released. The IRS fraud scandal raised another issue that still appears from time to time: Is it proper for the Federal government to turn known criminals into computer programmers? It is interesting to note that in a report following its investigation of the Leavenworth incident, a Congressional committee criticized the use of timesharing and advised against FPI involvement in systems that disburse funds, but approved of the program itself.[6]

TEACHING PROGRAMMING IN PRISONS

The basic argument against teaching computer programming is that it provides criminals with tools to commit more rewarding crimes, with less effort and risk than their previous adventures required. If the saying "once a criminal, always a criminal" were true, this argument would be unassailable. But it isn't; nearly two-thirds of all Federal releasees have been shown to be "successes."*

Because we are dealing with something so complex and unpredictable as human beings, it is impossible to say just why some releasees succeed in remaining out of prison while others fail. But we have clues. We know that people who commit crimes have the same basic desires as the rest of us; security, status, and so forth. The main difference seems to be that the criminal is more impatient, wanting his desires met immediately. And they often lack the training to fulfill their desires through legitimate means.

The FPI data processing program is a long-term, rigorous ordeal. An inmate must apply himself diligently for twelve months of training to avoid being dropped from the course. After completing the course, he has to put in two or three

* From "Success and Failure of Federal Offenders Released in 1970," A Bureau of Prisons study which defines a "success" as having no parole revocation or new sentence of 60 days or more.

years of further training in live production work, steadily improving his skills, in order to land a job upon release. Many never make it through the first few months of training.

Earlier release is not the motivating factor for these participants. The Parole Board gives no consideration to the programs an inmate completes. But, over the years, inmates have watched our participants released to well-paying, interesting jobs, and they hear through correspondence about the satisfaction these people get out of their new lives. This is the predominant reason given for wanting to participate in our program.

The program itself, then, is self-screening. We feel it attracts and keeps only the inmate who has made a firm commitment to play by the rules. The attitude of wanting instant, effortless success would not carry a trainee halfway through the first course. It is difficult to imagine someone still caught up in the syndrome of instant gratification having the patience to undergo such a grueling ordeal.

We feel the program is worthwhile because it provides an opportunity for those suited for a data processing career, who otherwise might continue robbing banks, or stealing securities to change their behavior. We have clear indications, after six years of operation, that our successful participants do go on to lead productive lives.

While it is difficult under existing procedures to follow up individuals released from custody, we can tell from the FBI's National Crime Information Center Computerized Criminal History File whether a releasee commits a new offense. From a continual review of these records, we know that no former participant has ever been charged with a computer crime. Moreover, these records indicate that 85 percent of our successful participants have had no further involvement of any kind with the law.

Even sophisticated statistical studies (which the above is not) dealing with the issue of relating a releasee's rehabilitation to his custodial treatment are subject to attack. There are simply too many factors that account for human behavior. Because the data, as positive as it appears, is inconclusive, one has to look elsewhere for additional indications that the society is better served by continuing the program. One such indicator is the attitude of private firms who hire our "graduates."

I hesitate to list these firms for fear of possibly embarrassing ex-inmates now leading productive lives in their employment. Key officials in these companies are of course aware of the men's backgrounds, but peers in some cases may not be. These companies would be recognized immediately by name. They are national and international in scope and have become prominent through hardnosed decision-making. These firms have decided that hiring ex-inmates as computer professionals is good business, and send the same teams that conduct campus interviews to our "campus." Recurrent hiring by the same firm is not uncommon.

It seems, in the final analysis, that judging whether teaching inmates to program computers is wise is, and should be, in the hands of these companies that hire our releasees. It is they, after all, who stand to lose if our critics are right. If they felt that the program was dangerous, they could stop its effects, and the program itself, by simply refusing to hire participants. Instead, they urge us to continue; one firm tells us they get more from our experienced people than from fresh college graduates.

Computer programming in the Federal institutions will therefore go on. Our internal security procedures, and the attitudes of the inmate participants, are adequate to prevent inmates' misusing of this knowledge while in our custody. Because private sector employers have confidence in the program and its graduates, the data processing community as a whole should accept these individuals, not as criminals with new tools, but former offenders making a useful contribution to society.

REFERENCES

1. "Guidelines for Automatic Data Processing Physical Security and Risk Management," *Federal Information Processing Standards Publication 31,* U.S. Department of Commerce/National Bureau of Standards, June, 1974.
2. Teibholz, Stephan W. and Lewis D. Wilson, *Users' Guide to Computer Crime—its Commission, Detection, and Prevention,* Chilton Book Company, 1974.
3. Baker, F. T. and Harlan Mills, "Chief Programmer Teams," *Datamation,* December, 1973, pp. 58-61.
4. Burke, E. L., M. Gasser and D. W. Lambert, "Federal Prison Industries ADP Security Study," MITRE Final Report, November, 1977.
5. "Manual for Security in ADP Systems," Department of Defense Directive 5200.28-M.
6. "Staff Study of Computer Security in Federal Programs," Committee on Government Operations, U.S. Senate, February, 1977.

APPENDIX—*CURRICULUM*

The one-year training phase is designed around behavior/skill producing modules which permit:

Flexibility in teaching and learning in a penitentiary;
Frequently measured goals which culminate in the PTU objectives, and,
Accurately defined functional areas of computer programming technology.

Thirteen modules are established as follows (each four weeks long, 160 classroom hours) with the objectives as stated.

Computer Programming Voctational Training by Behavioral Module

1. *The Computer and its Program, People, and Peripherals.* An introduction to computer programming as a technical science.
2. *The Scientific Programmer.* Learning and applying the FORTRAN programming language.
3. *The Business Programmer.* Learning and applying the COBOL programming language up through the use of tables and index-sequential files.
4. *The Systems Programmer.* Learning and applying the NEAT/3, Level 1 programming language.
5. *Operating a Computer for Fun and Profit.* Computer operations.
6. *The Computer Utilities and Maintenance Men.* The use of software utilities to perform discrete jobs, and the modification of existing computer programs.

7. *All De-Buggers are not Household Insect Sprays.* The art of writing a computer program from specification to completed program, without waste or futility.
8. *The Super Business Programmer.* Advanced COBOL through internal sorts, conversion to IBM from NCR, and beginning JCL.
9. *NCR versus IBM—How Does it work?* IBM utilities and advanced JCL and relationships to NCR's ways.
10. *Garbage In, Garbage Out.* Designing data systems, and test data.
11. *Critical Issues Seminar in Computer Science.* Discussion of current developments in the field, i.e., computer security, information privacy, etc. Use of outside guest lecturers such as NCR/IBM representatives, prior releasees, representatives from users organizations.
12. *I think you heard what I meant, but I'm not sure I said what you think you heard!* Documenting a computer program and current documentation standards required in the production phase.
13. *Where do we go from here?* Student evaluation. This module broken into four parts for use by instructors to evaluate students at 3, 6, and 9 month intervals, and at the end of the course of instruction for acceptance in the production phase.

Computer security differences for accidental and intentionally caused losses*

by DONN B. PARKER
SRI International
Menlo Park, California

INTRODUCTION

Research papers generally treat the problem of computer security as an intentional act to be solved by intellectually challenging methods such as encryption, descriptor-based systems, and program proving. Accidental and intentional acts are almost always referred to simultaneously; however, the safeguards are designed strictly to prevent intentional acts. It is generally thought that the same safeguards and strategies will be effective against both accidental and intentional acts. This is not necessarily true.

In a paper on security assessment by Glaseman, Turn and Gaines,[1] accidental and intentional differences are not stated but are made subservient to the question of exploitation of vulnerabilities. As with most research papers on security, important research issues are discussed but they are oriented principally to intentional acts. This is evident in use of terms such as "intruder," "attacks," and "exploit," but no discussion penetrates research needs for errors, omissions and other accidents.

In contrast, Courtney states in a paper on risk assessment:[2]

> *It is important that proper weight be given to the impact of errors and omissions. Data is more often destroyed or otherwise rendered useless or even harmful by people making mistakes than through dishonesty or malice. People whose loyalty and honesty are unquestioned but whose judgment and competence leave much to be desired are our greatest enemies. Data security consideration must not be limited to concern for the acts of dishonest people. Otherwise, it is very difficult to achieve proper cost justification of appropriate security measures. Weigh all potential security problems. . . The differences between accidentally and intentionally caused losses are many and profound.*

Unfortunately, he does not pursue these strong position statements to exploit the differences.

* The preparation of this paper was supported, in part, by the National Science Foundation Grant #MCS76-01242. However, any views or conclusions in this paper should not be interpreted as representing the official position or policy of the sponsor or SRI International.

The differences between accidental and intentional acts are profound. Accordingly, the requirements for the strategies and implementation of computer safeguards should respond specifically to each kind of act. Nevertheless, most research stops short of considering two security problems. It is therefore suggested that a more successful approach to the problem would be to regard the accidental and intentional acts as distinct entities requiring separate solutions. Treating each act separately would make the problem-solving process more manageable and more explicit so that combining the two derived strategies and safeguard applications will result in a more effective approach to both aspects of computer security.

The first step is to identify characteristics of the two problems. Each of the characteristics is discussed below.

CHARACTERISTICS OF ACCIDENTAL AND INTENTIONAL ACTS

Frequency

Incidence of errors and omissions is usually sufficiently high to support statistical analysis of loss cases in a computer service organization. A limited number of cases, however, is often tolerated because the expense of reducing incidence any further is not cost-effective. Analysis can be used to identify the major locations and sources of errors and omissions. Statistics on the incidence of accidents can be used to calculate the probability and expected loss from these problems.

Conversely, the low incidence of intentionally caused losses makes statistical analysis difficult. Only in limited cases, such as credit card and some other payment frauds, has the incidence been high enough to be subject to statistical analysis.[3] Most intentional acts, especially those involving significant losses, occur so infrequently that attempts to calculate probability, expected loss, nature, patterns of location, and source, are difficult if not impossible. Therefore, risk analysis at the present time must be more subjective and theoretical. The experience of victims that could be similar to experiences of future or potential

victims may be helpful in calculating statistical probabilities, but will probably not be sufficient.

Unsolved problems

There are few unsolved problems with regard to accidentally caused losses. Efforts to control unintentional losses have been made throughout the development of electronic data processing. Adequate solutions are known for most common errors and omissions. The causes of errors and omissions are mostly obvious and the solutions equally obvious. It remains to apply these solutions in a cost-effective manner. Nonetheless, despite the errors, computer operating systems, for example, are still effective.

In contrast, significant problems in security against intentionally caused losses remain unsolved. The technical security of currently available computer systems is inadequate against attacks from potential perpetrators (such as systems programmers) with sufficient skills, knowledge, and access to the systems. Moreover, computer operating systems are basically unpredictable. Therefore, proving the integrity of a system and guaranteeing freedom from unauthorized changes or enhancements are not possible at this time. Neither practical prevention nor adequate detection capabilities are currently known for such sophisticated attacks as Trojan Horse methods, salami techniques, data leakage, logic bombs, asynchronous attacks, or post system failure attacks.[4] Another problem yet to be solved is the adequate identification verification of legitimate remote system users. Where technical safeguards are inadequate, therefore, more difficult and costly methods of safeguarding must be developed for protection from intentional acts.

Act complexity

Accidentally caused losses derive from single, isolated acts. Each loss incident is usually independent of other loss incidents. On the other hand, an intentionally caused loss can result from sequences of dependent and independent authorized and unauthorized acts. The loss can occur from intentional efforts that capitalize on observed errors and omissions. Perpetrators often increase the complexity of intentional acts as a means of avoiding detection or apprehension.

Singularity of source

One person usually is responsible for an error or omission even though other people may cause extenuations, whereas more than one individual frequently perpetrates an intentional act. Half the known cases of computer abuse have involved collusion. Compared to manual, or non-computer related fraud, embezzlement, etc., there is much more collusion in perpetrating these crimes via computer. The reason for the high degree of collusion is that computer crime requires more skills, knowledge, access, time or resources than any one person usually possesses in the technically oriented environments of the computer systems. Security strategies and safeguards must be far more elaborate when one attempts to deal with the possibility of collusion.

Complexity of perpetrator behavior

Behavior of people causing errors and omissions is relatively simple. The behavior is related to the conditions at the instant of the act. After the act, the perpetrator need, at most, defend only his weakness that resulted in error. In contrast, the behavior of people performing intentional acts is often highly complex. Interviews with 23 people who intentionally perpetrated a computer crime revealed that these people had personal problems to solve or goals to reach preceding their search for vulnerabilities of a computer system or work environment.[3] Searching or studying a system for vulnerabilities is, however, only one facet of the complex behavior pattern of intentional perpetrators. Once motivated to penetrate and use a system to his own ends, a perpetrator plans, plots, gathers information, organizes, and conspires, and is left finally to rationalize all of his intentional acts. Effective security must address all of these issues.

Sources of security assistance

Accidentally caused losses have been extensively studied and reported in the technical literature for many years. As stated previously, however, most technical literature that claims to address both accidental and intentional acts effectively addresses only accidentally caused losses. Professional societies and governmental organizations dedicated to the prevention of intentionally caused losses have only recently attempted to examine the problems associated with computer technology. The American Society for Industrial Security, the U.S. Justice Department, the International Association of Chiefs of Police, National Crime Prevention Association, and Surety Association of America have barely, if at all, seriously or significantly addressed the criminal aspects of computer usage.

Security checklists

Much security literature relies heavily on checklist or cookbook approaches to computer security.[5,6] Strategy is based on the concept of implementing the well-known safeguards and controls identified in the checklists. Although this approach is particularly effective for handling errors and omissions, it is neither sufficient nor effective in dealing with intentionally caused losses. Perpetrators are aware of the safeguards and controls identified in checklists, thus their methods for compromising the system are designed to avoid them. Merely installing a named safeguard in a checklist is not necessarily sufficient protection from intentional acts. Moreover, the checklist rarely includes information

concerning the need and means of protecting the safeguard itself from attack or compromise.

Strategy and safeguard independence

Each error or omission possibility can be effectively treated in isolated ways, because one loss will be independent from any other loss. Thus, a safeguard against an accidental loss can be implemented without considering other possible losses. Basically, the formula of matching one error with one safeguard is reliable.

On the other hand, each intentionally caused loss possibility must be covered comprehensively, including all possible acts leading to the loss. A safeguard against an intentional act will often have impact on other types of acts or losses in that it may only partially supply adequate protection. A particular loss could result in a security strategy involving many safeguards or a defensive depth of multiple rings of safeguards which may be a desirable strategy. Further, the implications of imposing a safeguard must be considered in terms of its potential to open new possibilities of vulnerabilities. If a particular safeguard temporarily deters a perpetrator, he will simply seek ways around it or other vulnerabilities.

Safeguard compromise

In accidental loss situations, it can generally be assumed that the integrity of the safeguard will be ensured and sustained. Safeguards may be subject to failures from errors, but in general the errors will be independent of the errors that the safeguard is preventing. In contrast, safeguards are subject to intentional attack as part of the perpetrator's strategy in carrying out an intentional act. The compromise of safeguards becomes part of the total loss event that the safeguards are meant to prevent or detect. In this sense, a safeguard must be considered as an asset subject to protection if it is to perform adequately when needed.

Level of protection

Determining the level of protection that a safeguard from an accidental act affords is simple because of the one-for-one, act-safeguard relationship. Higher incidence of accidents can result in more easily calculated protection levels. In intentionally caused acts, the level of protection is determined by a combination of security measures associated with a given asset. Lack of statistics makes effectiveness difficult to determine.

Achievable protection level

Security from accidental losses can be achieved to a high degree or, at least, can be easily controlled at a wide range of loss levels. One reason for this is that the success of protection can be accurately measured from incidence statistics, and the security then can be changed on the basis of the need that is determined. Because there are few unsolved protection problems with accidental losses, it is possible to control incidents to any degree desired within the limits of economic considerations.

A high level of protection from intentionally caused losses is difficult to achieve because of incomplete knowledge of all vulnerabilities. An individual loss could be affected by many factors, including the difficult-to-determine completeness of protection. The one-upmanship escalation of protection and increasing sophistication of perpetrators' methods precludes reliable measurement.

Potential perpetrators

The population of potential perpetrators of accidental loss is easily identified. It consists of those persons having the access and authorization to perform an act that might result in an accidental loss. Conversely, the population of potential perpetrators of intentionally caused losses is difficult to identify. It includes not only those identified relative to accidentally caused losses, but also the often larger numbers of people who might gain the necessary skills, knowledge, and access to perform an unauthorized act. For example, in remote terminal usage, impersonation alone would account for a large number of people who might possess such skills to replace people in positions of trust.

Potential perpetrator capabilities

In accidentally caused losses, the minimum skills, knowledge, and resources of potential perpetrators are at issue. Access is a prerequisite. Beyond this, the security specialist is dealing with the minimum skills and knowledge of people that are the source of errors and omissions. With intentionally caused losses, the security specialist is dealing with people who potentially have the maximum possible skills, knowledge, access authorization, resources, and time to commit the act. The attack possibilities and matching protection activities constitute a game in which each side is pitting its maximum capabilities against the other side.

Loss limits

Accidentally caused losses are limited in size. Where the size of the loss can vary, the probability of detection and termination of the loss grows rapidly with the size of the loss. The victim is ordinarily able to observe, measure, and limit accidental losses in a timely manner. However, detection of intentionally caused losses in a timely manner is not necessarily related to the size of the loss. Exceptionally large losses can go undetected when the perpetrator puts forth significant effort to conceal the losses. Also, in increasingly automated criminal methods, the possibility of a large loss in a time scale measured in milliseconds before humans can react becomes more likely.

Detection

Perpetrators of accidentally caused losses have no conscious intention to err before and during their acts. Therefore, avoidance of detection occurs, if at all, after the act has occurred. There is generally less fear of detection or of reporting the loss and more cooperation with the victim in recovery in accidental loss situations as compared to the perpetrators' position relative to intentional acts. Interviews have revealed that perpetrators greatly fear unanticipated detection before, during, and after their acts, and often much of their efforts and resources go into prevention of detection. This makes detection of intentionally caused losses a much greater challenge than for accidentally caused losses and should occupy a greater amount of the security specialist's attention.

Theoretical approach

Support for claiming a difference in risk between accidental and intentional acts can be shown in mathematical terms, even though this procedure is highly theoretical and not intended for practical use. Given are an asset subject to accidentally or intentionally caused loss, its environment, and a population of potential perpetrators with known skills, knowledge, access, resources, time and motivation. The probability P of a particular type and size of loss L from any possible independent accidental acts a_i $(i=1,2,\ldots n)$ with probabilities of their occurrence $p(a_i)$ is $P(L)=\Sigma_n p(a_i)-\Pi_n p(a_i)$. This suggests that a safeguard that reduces the probability of an act a_i for any value of i will reduce $P(L)$.

The probability P of a particular type and size of loss L from any possible independent intentional acts a_i $(i=1,2,\ldots n)$ with probabilities of their occurrence $p(a_i)$ is $P(L)=\text{Max}_n\{p(a_i)\}$. This conclusion is based on the theoretical premise that any potential perpetrator will perform rationally in choosing the one act with the greatest probability of success. This suggests that an entirely different result would occur from installing a safeguard that reduces the probability of an intentional act a_i for any i. If it were the act with greatest probability, it would have the most significant effect on $P(L)$ by directly reducing it by the amount that would cause the act of next largest probability to determine the value of $P(L)$. If it were any other act that was affected, there would be no change in $P(L)$. Therefore, installing safeguards has no effect except for those that reduce the probability of the act with greatest probability of occurrence.

Two different security strategies are suggested by these theoretical conclusions. Optimum strategy for security from accidentally caused losses is to reduce the probabilities of that combination of acts that reduce $P(L)$ by the greatest amount to an acceptable level, given limited security resources. Optimum strategy for security from intentional acts that reduce $P(L)$ to an acceptable level is to reduce the probability of the act with maximum probability followed in sequence by reducing the probability of subsequently maximum probability acts until security resources have been expended or $P(L)$ is reduced to an acceptable level, whichever occurs first.

Applying any safeguard that reduces the probability of any accidental act will incrementally increase security. However, applying a safeguard that reduces the probability of an intentional act may not increase security unless it reduces the probability of the one act that represents the maximum probability of occurrence. Thus, security from intentional acts is no greater than the weakest link. This shows a major difference in the two kinds of problems.

The most obvious weakness in this argument is the assumption that the population of potential perpetrators is rational in always selecting the act with the greatest possibility of success. In addition, practical application of the theory implies that the population and all acts relative to loss of a particular asset are known.

Applying a strategy only for accident prevention or only for intentional act prevention would clearly result in suboptimization, because many safeguards, some with minor modification, serve both purposes. A combined strategy, although not fully optimized, offers significant advantages over the near random-based, currently used strategies that are organized around merely computer center functional approaches (physical, operational, procedural and system) to security strategy. The steps to be followed in applying a combined strategy are:

(1) Select the intentional act with largest probability of occurrence.
(2) Apply safeguards to reduce the probability.
(3) Identify the accidental acts also reduced in probability of occurrence.
(4) Repeat Steps (1), (2), and (3) with the newly identified intentional act with largest probability until probability of intentionally caused loss is reduced to an acceptable level.
(5) If the probability of accidentally caused loss is still not acceptably lowered, apply safeguards to reduce the probabilities of the combination of acts that reduce the probability of accidental loss to an acceptable level.

The alternatives of applying safeguards against accidental loss first or applying safeguards without regard to accidental or intentional act differences are less effective. Safeguards aimed only at accidental acts often will be ineffective or will not be the best safeguards against intentional acts. One reason for this is that the safeguards may be installed only with the intent of protecting against the compromise of assets and not with the intent of protecting the safeguards themselves against compromise. In addition, a particular order of implementation of safeguards may not be responsive to the strategies of the potential perpetrators who are looking for the simplest and safest route to achieving success.

Failure to use the recommended strategy results in the Maginot Line Syndrome: A monolith of obvious and traditional safeguards will be installed, and the perpetrator

merely bypasses it by finding the weakest link not yet addressed by the victim organization.

ILLUSTRATION

Implementation of access control by password to an on-line system from a terminal illustrates the problem. For purely accidental access prevention in a benign environment, identification verification by input of a minimal length password or name is all that is necessary. Prevention of intentional, unauthorized access to the system requires the following additional measures:

- Sufficient password length to reduce exhaustive search attempts.
- Assignment of randomly generated passwords to reduce guessing by analysis of dependence between password content and password holder characteristics or knowledge.
- Nonvisible input of passwords to avoid observation or scavenging.
- Frequent briefings of password holders concerning safekeeping of passwords.
- Safe password administration, including separation of duties; invocation of need-to-know principle; appropriate transport and storage of passwords; and specific authorization of duties to accountable and trusted employees to avoid spoofing, coercion, degradation of care, and unauthorized physical access attempts.
- Encryption of on-line system password files and password comparison in encrypted form in privileged mode computer operation to prevent technical compromise.
- Limit guessing attempts of unauthorized passwords by imposing time delays between input attempts and disconnect after n failed attempts.
- Record and verify password use periodically with password users to avoid password theft and unauthorized use.
- Analyze patterns of password use and failed access attempts for deviations from normal experience to detect unanticipated attacks.
- Test all protection mechanisms frequently and prepare and follow contingency plans to avoid attacks when system or operational failures occur.
- Perform frequent, random, independent, and visible audits to inhibit potential perpetrators and ensure safeguard integrity.
- Impose sanctions against violators.

These additional safeguards are not new, but it is difficult to find them all stated in a single source. Again, subsets of them have been implemented, but are not generally found in a single computer service organization. For prevention of accidental access, only a few are needed. For prevention of intentional acts, all are needed; however, total implementation of all safeguards still is insufficient protection in many high-risk environments. Many more detailed specifications are needed for such items as levels of encryption effectiveness, pattern analysis of password use, password length, and audit techniques.

CONCLUSION

Glaseman, Turn and Gaines conclude by stating two major requirements for progress in security assessment:

(1) Increased research aimed at the development of a better understanding of the informational elements of security assessment, and
(2) Experience at the level of individual computer installations, in the application of a broader and more accurate information base to the assessment of computer security.

A third requirement should be appended to those suggested above:

> Recognition and separate treatment of two separate and distinct computer security problems, accidentally caused losses and intentionally caused losses.

REFERENCES

1. Glaseman, S., R. Turn, and R. S. Gaines. "Problem Areas in Computer Security Assessment," *Proceedings NCC 1977 Vol. 46,* pp. 105-111, AFIPS Press, Montvale, N.J.
2. Courtney, R. H., Jr., "Security Risk Assessment in Electronic Data Processing Systems," *Proceedings NCC 1977 Vol. 46,* pp. 97-104, AFIPS Press, Montvale, N.J.
3. Parker, D. B. "Computer Abuse Perpetrators and Vulnerabilities of Computer Systems," *Proceedings NCC 1976 Vol. 45,* AFIPS Press, Montvale, N.J.
4. Denning, D. E., and P. J. Denning, "The Limits of Data Security," *Abacus (Mock-up), Vol. 0, No. 0, June 1977,* pp. 22-30. AFIPS Press, Montvale, N.J.
5. Martin, J., *Security, Accuracy and Privacy in Computer Systems,* Prentice Hall, New York, 1973.
6. Patrick, R., *System Review Manual on Security,* AFIPS Press, Montvale, N.J., 1974.

Anatomy of a computer crime*

by SUSAN HUBBELL NYCUM
Chickering & Gregory
San Francisco, California

INTRODUCTION

This paper describes the process of an investigation** into the technical and legal aspects of a computer abuse of alleged theft of programs stored in a computer. The goals of such investigation are to provide information to help make computers and computer systems more secure, to provide computer technologists and managers with data as to the potential exposure for a particular activity, and to provide law enforcement officers and prosecutors knowledge of whether and to what extent existing laws may apply to the case.

There is no wish on the part of the investigators to sensationalize a case or bring to the victims and perpetrators more notoriety or exposure than already exists from public documents and proceedings or prior press coverage. Therefore, the paper will not use actual names and will where possible, generalize rather than particularize places and organizations.

Methodology of investigation

To date the investigators have knowledge of over 500 reported cases of computer abuse. To investigate each of these is beyond the existing level of resources. The cases that are selected for field study are those which offer one or more of the following:

(1) a plethora of available detail;
(2) an interesting, yet typical rather than wildly bizarre modus operandi;
(3) a suspected or demonstrated significant gain to the perpetrator or similar loss or exposure to the victim;
(4) a nontrivial legal question as to what crime or legal wrong has been committed; and
(5) a nontrivial investigative question for the law enforcement agency.

These five topics are more fully developed as follows:

(1) The incident must be available for research. The project prefers to investigate an incident at the postresolution stage. This qualification more readily assures that the parties will candidly share their experiences. While individuals tend to rationalize their actions at all stages of an incident, postconviction interviews with perpetrators are more candid than pretrial or presentencing interviews. Similarly prosecutors and law enforcement personnel are understandably constrained from discussing a case before conclusion. Victims considering or in the course of civil litigation are normally uncommunicative.*

All aspects of the incident must be open to research to avoid the three common barriers to objectivity: bias, prejudice and interest of the parties thereto. In addition, each person involved sees the incident from a different perspective. As with the old anecdote about the blindfolded group asked to describe an elephant by touch, one at the trunk, one at the tail, one at the feet, each party to an incident "sees" the case somewhat differently and extrapolates the whole from his own experience. The investigator needs to put those various viewpoints together and can best do so from a wide variety of interviews.

(2) There are always unique approaches to the perpetration of computer abuses and there are always "old friends" of tried and true approaches. Since the project has six years of experience, many of the typical approaches have previously been identified, investigated and catalogued. If an incident looks like a good example of a known modus operandi, there is low utility, absent extenuating circumstances, in investigating that type again. At the same time a "far out" approach, dependent on a set of unique circumstances has lower utility than an abuse which could happen repeatedly to a larger set of victims, e.g., point of sale terminal users. This is not to say that new twists to old abuses or sophisticated efforts which thwart existing security are ignored. These may be the most critical cases from which to learn improved security techniques.

(3) The project does not chase the illicit makers of a

* The preparation of this paper was supported, in part, by the National Science Foundation Grant #MCS76-01242. However, any views or conclusions in this paper should not be interpreted as representing the official position or policy of the sponsor or Chickering & Gregory.
** As performed by Susan Nycum and Donn Parker pursuant to the above National Science Foundation Grant.

* However, many victims and perpetrators have given information to the project in confidence and that confidence is carefully guarded by the investigators.

snoopy calendar* any more than a traditional investigator is concerned with the typical business practice of taking pencils from the office for home use. The project prefers to spend time where the results of the study will help minimize exposure to nontrivial risks.

(4) From the standpoint of legal research, abuses to software and services are of greater significance than those to hardware. This is because hardware, however complex, is visible, touchable material, the generic equivalent of other materials the abuses to which the law has coped with for centuries. Software and computer services, e.g., computer time, are new forms of intangible property and less amenable to traditional legal treatment.

(5) How does one find out in what way the abuse happened, how does one prove what happened for the purpose of prosecuting the act and later implementing security against similar occurrences and implementing proper audit trails. This criteria can be evaluated with respect to inventiveness of the modus operandi or with respect to the rules related to evidence gathering, e.g., telephone traces.

Once an incident is selected for investigation, the project contacts the cognizant persons, the victim, perpetrator(s), law enforcement persons: prosecutors, and judicial personnel and sets up *personal* appointments with these people as nearly contiguous in time as possible at the interviewee's office or residence (including jail cells where applicable). (This helps to preserve the continuity of the investigation and to assure that similar questions and similar weighting techniques as to response are used.)

At an interview the researcher explains the project goal and invites a description of the event in the interviewee's own words. The approach is one of openness and professional interest. There is no attempt to intimidate or lead the interviewee. There is also a minimum level of sympathy or support for action taken. Experience indicates that rapport can be established with the interviewees without compromise to the interview—the project is not on one side or the other. New points that arise in a later interview are checked for comment by earlier interviewees on a call back basis. There are patterns to interviews but these vary with the type of abuse and the class of interviewee, e.g. victim, prosecutor, perpetrator. Notes of interviews are transcribed as soon as possible after the fact and details of the investigation rechecked and gaps filled in and authorities explained by subsequent contacts.

Results of case studies are added to an aggregate of data on several issues, project findings are never based on one incident.

*Selection of the Brown** case for investigation*

The Brown case was a theft of proprietary software from a private installation over telephone lines. It was investigated by the FBI, prosecuted by a U.S. attorney and the defendant pled not guilty. At a resulting jury trial the defendant was convicted on two counts and one count was dismissed. The facts were typical of a particular form of abuse which had not been previously investigated in depth. There were interesting legal and investigative issues and the motive of the perpetrator was in dispute at the trial. All parties were available for interview.

THE FACTS OF THE BROWN CASE

The facts of this incident have been undisputed since the perpetration was discovered and the perpetrator apprehended. The main and perhaps only contention is as to why the act was done.

The parties agree that the perpetration consisted of the unauthorized copying of a significant portion of a systems program from the installation on which it was running by means of remote terminal access over dial-up telephone lines.

The perpetrator ("Brown") was a former employee who had had supervisory status for technical conversion and operation of a data center, at the victim company. Brown had executed a confidential information and invention agreement at the time of hire which stated in relevant part:

> "I will not disclose to anyone outside of [Company] or use in other than [Company's] business any confidential information, information or material, including but not limited to any and all information, material, data or memoranda relating directly or indirectly to any technique, tool or method used and/or applied by [Company] in any computer program or project related to the business of [Company] or its subsidiaries, either during or after my [Company] employment except with [Company's] written permission."*

All the accessing of the system and copying of the program was done after Brown left the company.

The program copied was a heavily modified variant of a system originally developed by an educational institution and financed in whole or in part by government grants to that educational institution. No data was copied and no other program was copied.

Detection

The perpetration was originally discovered by an alert employee who noticed that an imposter was signed onto the system. The detection of the presence of an imposter was possible because the person whose identification was being used was in sight of the alert employee and not engaged in a terminal session. System capabilities were used to monitor the terminal session and telephone company traces specified the origin of the terminal use.

---

* Note, however, that S. 1766, The Federal Computer Systems Protection Act of 1977, makes such activity susceptible to a $50,000 fine or 15-year prison sentence.

** An actual case with names changed.

* Official Transcript, pp. 27-28 of Proceedings.

Prosecution

As the telephone access was across state lines, the FBI investigated the incident and the cognizant U.S. attorney prosecuted the perpetrator for violation of federal law, Section 1343 of Title 18 of the United States Code (wire fraud).* The indictment was summarized by the judge as follows:

> "That the defendant, [*Brown*], obtained the unpublished telephone access number to the computer systems operated at and by [Company] exclusively for [Government];
>
> That the defendant unlawfully and without authority obtained account numbers, user identification numbers, and other information which was necessary in order for him to access the said computer and computer systems;
>
> That the defendant, through the use of said information, gained access to the computer systems operated by [Company] for the exclusive use of [Government], including the system known as [X];
>
> That the defendant without authorization used said identification numbers and account numbers, and without authorization obtained information, including print-outs, from the computers and computer systems, then knowing that he did not have authorization to gain such access and to retrieve such information.
>
> . . . It is further alleged in each of Count I and II that for the purpose of executing said scheme to defraud, the defendant caused to be transmitted in interstate commerce by means of a wire communication, namely a telephone communication between the defendant in [Y State] and [Company in Z State], certain signs, signals and sounds. Count I alleges a transmission on our about _____, 1975 and Count II alleges a transmission on or about _____, 1975.
>
> In summary, then, the defendant is charged with devising a scheme to defraud and with using a wire transmission in furtherance of that scheme. The same scheme is involved in each count, although different transmission have been alleged."*

A third count which alleged receipt of stolen property, was dismissed by the judge who ruled that the electronic impulses transmitted by telephone line across state lines could not constitute stolen goods because they were not tangible items.

The defendant admitted the act of copying the system but argued first that the system was in the public domain and thus he hadn't taken anything proprietary from an owner and second that his intention was not to use the system for his own interests but to show the vulnerability of the company's systems security.

After a two and one-half hour deliberation, the jury found the defendant guilty on both counts.

Similarity to other cases

The fact pattern of this incident was close to that of an abuse which took place in California several years earlier. In the former case a programmer pled guilty to theft of trade secrets. These trade secrets consisted of a proprietary application program the perpetrator had copied over remote telephone lines to his employer's computer and then printed out a hard copy thereof on his employer's printer. The perpetrator in that case admitted that he intended to use the program for competition on behalf of his employer against the victim.

The contention of the perpetrator in the Brown case that his intention when he copied the program was to show the Company that its systems security was inadequate. This is most reminiscent of a 1974 Oregon case of computer abuse in which a student notified the console operator that he had taken control of the computer operating system from a terminal and had done so for the purpose of pointing out the security vulnerability of the system. The system in the Oregon case stored motor vehicle registration data including personal information. The system in the Brown case was targeted to contain government data classified "secret."

UTILITY OF THE CASE TO THE STUDY

The case was of great utility to the study of computer abuse from several aspects.

Security implications

The victim company described its systems access controls as a three part key number access: account number, associated user identification and password all of which had to be provided by the user and verified by the system before computer time could be used and files could be accessed. In their interview with the study the spokesmen were candid about their concern for the security breach but indicated the level of existing security was adequate to protect from external intrusion. Another employee of the company stated in a separate interview that steps had been taken after the incident to tighten the level of security. Testimony at trial indicated a security weakness in that passwords were infrequently changed:

> Q. How often are the passwords changed?
> A. Right now, they are changing once a week.
> Q. Well, up until the time of this little problem, how often were they being changed?
> A. At that particular time, I couldn't tell you, but it was not very often.
> Q. Well, I guess the next question obviously is, has the password been changed periodically as a result of this incident?
> A. They were supposed to be changed periodically before the incident.

* §1343. Fraud by wire, radio, or television. Whoever, having devised or intending to devise any scheme or artifice to defraud, or for obtaining money or property by means of false or fraudulent pretenses, representations, or promises, transmits or causes to be transmitted by means of wire, radio, or television communication in interstate or foreign commerce, any writings, signs, signals, pictures, or sounds for the purpose of executing such scheme or artifice, shall be fined not more than $1,000 or imprisoned not more than five years, or both. Added July 16, 1952, c. 879, §18(a), 66 Stat. 722, and amended July 11, 1956, c. 561, 70 Stat. 523.

** Transcript pp. 832-833.

Q. Well, I think that's clear, but I mean did this incident bring on a policy of changing those passwords?
A. No, the policy was there prior to that.
Q. Did this incident bring on an implementation of the policy of changing passwords?
A. Yes, it did change the password, yes.
Q. Okay, because they found that to be a weakness in the system, isn't that fair to say?
A. That's correct.
Q. And how about the identification numbers now, are they being changed more frequently?
A. The systems people's identifications are being changed more frequently.
Q. Are they becoming more complex?
A. The format remains the same.*

The prosecutor

The prosecutor stated that this had been the first exposure of that office to computer abuse and they felt hampered by the lack of adequate federal law applicable to the case and by their own unfamiliarity with the type of activity. (The FBI could not be interviewed without a many months' delay because of backlog of requests under the Freedom of Information Act. Their reports, however, had been used by the U.S. attorney who did share his experience on the case with the study.)

Impact on programmers

From the standpoint of programmers involved with programs developed other than by themselves entirely with their own resources, the case points out two critical matters.

Most programs are owned by someone; very, very few are actually in the public domain. Indeed, another paper by the study is directed to a discussion of the ownership of programs whose development was sponsored in whole or in part by government funds. If the perpetrator Brown thought himself innocent of any wrongdoing when he took a copy of the company program, he was woefully mistaken. That assumption, which is unfortunately widespread, is wrong. Programmers must assume, unless assured otherwise by the persons responsible for the program, that it belongs to someone and permission to copy or use the program must be obtained in advance of such use.

System security is a legitimate concern of all those involved with computer installations and programs running thereon. If Brown was truthful, his efforts to convince the company during his employment of its lack of systems security through memo and meetings were unsuccessful. His testimony was that having failed to achieve better security while an er ployee of the company, Brown intended to copy the entire program after termination and present the company with the evidence of a series of perpetrations and the significant results of such accesses. (In fact he accessed the system approximately 60 times without authority.)* The jury disbelieved Brown. His case might have been stronger had those noble intentions been documented or had he in any way enunciated his concerns to users of the facility or to law enforcement personnel. Computer people whose personal ethics require action similar to Brown must realize and acknowledge that the action carries with it the possibility of conviction of a crime.

Legal aspects of the case

The case was interesting from a legal standpoint for several reasons.

The Judge's dismissal of the count of receipt of stolen property was the second time a jurist had been faced with the unauthorized transfer of a copy of a computer program in the form of electronic impulses over telephone lines. In both cases the judges found this act to be outside the purview of the penal laws affecting asportation (carrying away) of personal property.

The prosecutors felt hampered in their conduct of the case by the lack of directly applicable law. They were fortunate that the alleged theft had been interstate and thereby subject to federal jurisdiction and the federal wire fraud statute. Had the incident occurred entirely intrastate, the laws of the particular state involved might not have provided the source of a prosecutable offense.

The character of the property involved, software, exposed the prosecution and the defense to the complicated and confusing issue of identification and protection of proprietary interests in software. This issue, as discussed above, was also a source of stated misunderstanding by Brown.

From a security standpoint, the methods of perpetration and detection were of value in determining how to make computer systems more secure, both technically and operationally.

CONCLUSIONS

Conclusions reached from a study of this incident include:

1. There is a need to educate prosecutors and defense attorneys about computer abuse. This was not the first or only instance of compromise of a computer system and a theft of a program and it was not the only instance of such a compromise being made purportedly not for gain. It would have been helpful to each side to have known of other occurrences.
2. Prosecutors and law enforcement officers need information about the environment and terminology of computer installations so that investigation and case preparation can proceed more independently of the victim's guidance.
3. Computer organizations need to know how to protect proprietary software, legally and physically.

* Official Transcript, pp. 88-89.

* Transcript p. 749.

4. Computer professionals need to be aware of their legal exposure for appropriation for their own purposes of software not owned solely by themselves.
5. Computer organizations need to realize that security precautions are an ongoing concern and that procedures must be kept current to remain effective, particularly vis-a-vis current and former employees who know the system.

GENERAL REFERENCES

1. Nycum, S., "The Criminal Law Aspects of Computer Abuse: Applicability of the Federal Criminal Code to Computer Abuse," 5 *Rutgers Journal of Computers and the Law,* 1976, p. 297.
2. Nycum, S., "Legal Protection of Proprietary Rights of Software," *Southwestern Law Review* (in Press).
3. Parker, D. B., "Computer Abuse Perpetrators and Vulnerabilities of Computer Systems," *AFIPS Conference Proceedings, Vol. 45,* 1976, AFIPS Press, Montvale, N.J.

Federal policy and the future of computer communications services

by PHILIP S. NYBORG
Computer and Communications Industry Association
Arlington, Virginia

Communications policy has long been a matter of direct Federal concern, based on the government's responsibility to regulate the monopoly communications industry. As computer technology becomes increasingly reliant on communications services, and as computers become increasingly utilized in the provision of communications services, the Federal government is faced with difficult policy decisions in executing its mandate to regulate communications services while at the same time refraining from regulation and inhibition of developments in computer systems.

In the intersection of the computer and communications industries, there are major new markets emerging for data communications services as well as hardware which arguably has both data processing and communications functions. Equally important, other regulated services which have traditionally been provided in paper media, such as postal service and banking, are now shifting to electronic form as electronic mail and electronic fund transfers. The technological possibility of all digital networks, in which a wide variety of services including voice could be distributed in digital form under computer control, further complicates the problem of regulating monopoly communications carriers while at the same time maintaining a competitive free enterprise market in the nonregulated data processing field.

Incoming Federal Communications Commission (FCC) chairman, Charles D. Ferris, while affirming the FCC procompetitive posture toward the provision of certain communications services has recognized the scope of the new technological opportunities. Ferris was recently quoted in the *Washington Post* as saying "I don't think its Buck Rogers to conceive of a system in the very near future of homes and certainly businesses having not only voice communications with each other but access to data banks, even video communications" and citing the possibility of computer terminals in each home and business for transactions ranging from banking to ordering groceries.

The panel will address the present status of Federal policy, regulation and legislation impinging on computer communications, as well as relevant anticipated developments and their impact on the data processing field.

The discussion will commence with a brief overview of multiple regulatory jurisdictions within the Federal government which affect computer communications in their various forms, including the Federal Communications Commission, the Congress and groups within the Executive Branch. The panel will then focus on the major issues within these forums, beginning with consideration of issues currently addressed by the Executive Branch; of particular importance will be international communications, as well as the newly created office of the Assistant Secretary of Commerce for Communications and Information. Following will be two closely related presentations on regulation of the communications industry, as implemented by the FCC and the Congress. Discussion of the Federal Communications Commission will address the trend toward a pro-competitive policy and the distinction between communications and data processing which is being formulated in the ongoing FCC "Computer Inquiry" (Docket No. 20828). Current issues before the Congress will be discussed as an extension of recent FCC decisions, and particular attention will be given to the possible revision of the Communications Act of 1934 as well as other related legislation dealing with industry structure in the telecommunications field.

Area Director:
David C. Rine
Western Illinois University
Macomb, Illinois

Computing careers and education

The Technical Area of Computing Careers and Education offers those attending NCC-78 a state-of-the-art view of topics including Professional Requirements and Educational Alternatives, Accreditation of Information Processing Programs, Professionalism and Professional Development, Career Paths for Women, and Computers in Early Education. Viewpoints will be presented as to what educational skills and backgrounds recent graduates should have as they enter positions in industry and government. Needs will be identified for a formal evaluation program that will accredit the offerings of particular institutions and programs, similar to accreditation guidelines for computer engineering programs. Treatments of obsolescence problems through professional development for the computing professional, including guidelines for defining and conducting an individual professional development program, will be covered. It will, also, be postulated that women must more carefully design their careers in information processing. Finally, calculators and microcomputers are now cheap enough so that preschools, elementary schools and high schools can afford to give students quite good access to them, and better training and education with them.

A treatment—Professional development

by FRED A. GLUCKSON
National Bank of Detroit
Detroit, Michigan

"There's something going on here
but you don't know what it is . . ."
Bob Dylan

PREFACE

In the mid-1940's, the most advanced technology around was the Hydra-Matic transmission. The conventional wisdom among us kids was that the Hydra-Matic was so complex that no single person understood it all. A look at its valve body, which directed hydraulic pressure along various paths to effect gear changes, seemed to prove this. But the logical complexity of this valve was roughly the same as today's single-bit binary adder circuit. And such a circuit is an inconsequential component of today's most advanced technology—as exemplified by the Intel 8085 microprocessor, which has 6,200 transistors, can execute 770,000 instructions per second, and is housed on a chip .164 inch by .222 inch—smaller than Lincoln's head on a penny! In only 30 years the frontier of technology has surpassed anything we kids could have imagined. Most of us in this room will be alive in 30 more years. Where can the leading edge possibly be then?

Next, let us consider the career of Albert M. Diederich at the Ridge Tool Company. Mr. Diederich assembles wrenches. He has been assembling wrenches for 45 years. In that time, Mr. Diederich estimates that he has built 25,000,000 wrenches![1] He is, by all odds, the world's ranking expert at his trade; there is no problem beyond his knowledge or ability; no one can teach Mr. Diederich "a thing or two." Can anyone in this audience claim such expertise? Not likely! This is the nature of a career in information sciences: ours is a field in which experts disagree, where terms are ambiguous, but in which our achievements have been profound. We have used computers, in the Viking space program, to design and build a radio antenna so sensitive that it can perceive the energy radiated by a lighted match held on the surface of Mars.[2] We have designed a computer program that plays world-class chess and has beaten David Levy, an international chess master, at the computer's forte—speed chess—but not yet in regulation tournament chess. Mr. Levy described this program at ACM 77 in Seattle. "Chess 4.6 is fantastic. Nine years ago I wouldn't have believed it possible. It doesn't make tactical oversights. You can't fool it. But it can't make a long-term strategic plan. If it could, it would be a grandmaster."

Meanwhile, back in retail sales, a clerk waves a wand and the cash register prints the item description and price while inventory control systems a thousand miles away are notified of the transaction. In the classical sense, this is a magic wand. But for us, the data processors of today, the technology of such an application is well within our understanding. Or ought to be. Unlike Albert Diederich, we have chosen a profession that will leave each of us hopelessly behind if we do not search aggressively and persistently to stay with our art.

INTRODUCTION

It is my intention to suggest a program of self-development, aimed both at remedying known personal deficiencies and at keeping abreast of new applications of computing technology. There are many opportunities for self-development and I will point these out. But it remains for the individual computing professional to take the initiative to set up a program for himself or herself and to follow thru. It will require a commitment of time, and in some cases, money. But how could personal resources be better expended than as an investment in one's future?

One's career may be affected profoundly, as follows: employee appraisal forms usually have a section labelled "Job Competence". Other sections relate to personal traits such as "Leadership Skills" or "Interpersonal Relations". It is difficult for adults to change basic personality attributes such as leadership skills. So there is not a great deal that we can do about low ratings in non-technical areas. But we have no excuse for continuing to earn low ratings in technical knowledge.

When a computer professional takes a new job, his first performance appraisal may be lower than usual because of the new skill requirements. But having identified these deficiencies and established a program to remedy them, subsequent appraisals should improve. So there is little excuse for one's career progress being limited by technical inadequacy.

One large corporation strives to channel five percent of everyone's time into technical training. This comes to two

and one-half weeks a year, or two hours a week for formalized training apart from reading journals. If your organization is similarly enlightened, so much the better. If not, your five percent will have to come from personal time. Surely no one in this audience surrounds himself with TV and beer in free time. I suggest that you do not reduce the time spent with family—that is more important than career. Rather, you should defer some leisure time activities and substitute the pursuit of your own professional development goals. If this is too much of a burden, you may be in the wrong profession. The one resource each of us has in equal measure is time. Other facilities are not so evenly distributed. Personal wealth is not, nor is intelligence or good looks. But each of us has exactly 24 hours each day to be allocated as he sees fit. A major factor in success is one's ability to manage time to best advantage. The old cliche is: "If you need something done, ask a busy person to do it." You should not schedule yourself to be constantly busy. Relaxation is important to physical and mental health. But you should schedule your relaxation—not just let it happen. Do not drop in unexpectedly on a busy friend and expect a warm welcome.

Make a list of things to do and the priority of each. The first item on your list should be "make a list;" then when you finish the list you can immediately scratch off the first task and you have made progress.

OBJECTIVE

Perhaps an overall objective for self-development would be to help us distinguish good ideas from marginal ones. Not all ideas work; not all products are successful. Remember Corfam? Du Pont invested a king's ransom in this synthetic substitute for shoe leather. They ultimately wrote off this investment as a total loss. Does Viatron ring a bell? Ten years ago this company set about building a full-function terminal to rent for $21 per month. A good friend of mine, a computer professional, used pencil and slide rule (remember those?) to show that it could not be done. Viatron ultimately went into receivership. It takes years for the marketplace to render these judgments. A computer professional may be able to use intuition and analysis to render an informed appraisal. As of this writing the Commodore PET computer is described by its Project Manager[3] as selling for under $600, with volume projections of 5,000 per month. Service is to be handled by the retailer. Is this "viable"? What about conferencing by computer? Will this happen? Here is Dolotta on the subject:

> "We believe that over the next ten years the potential market for applications aimed at such 'offices of the future' may well equal the value of all of today's business data processing applications."[4]

Structured programming and distributed processing are current buzzwords. The computing professional ought to be well enough informed to have an understanding, and perhaps an opinion of these concepts. It is not sufficient to have written the world's tightest routine for converting 1401 five-digit operand addresses to properly zoned three-digit machine addresses. We're not doing this any more. So broaden and deepen your understanding so you can distinguish between good work and bad work in your sub-specialty of computing.

DEFINITIONS

Let's define our terms. We are talking about training, not education. Education is the business of elementary schools, secondary schools, and undergraduate liberal arts programs. In my opinion, the goal of education is to broaden the whole person, to develop logical thought processes, mental and physical discipline, communications skills, self confidence, broad awareness, and appreciation. In short, education prepares one for training. Without education, training can be self-limiting. Pre-med is education, medical school is training. Medical school prepares a doctor to diagnose illness or perform surgery; but pre-med enables him to grapple with concepts such as "health care delivery systems."

There is a growing trend in the lower schools that troubles me—the introduction of "computer labs" in secondary schools, and even in primary schools. Teachers are proud of turning their students on to computing at such tender years. No wonder the kids are excited and involved, and have to be peeled away from the terminals when the lab session is over. Programming and game playing are fun. Education is not. I was recently a guest lecturer to such a group of bright twelve-year-olds. They listened politely as I discussed how computers are used in banking—but when the compulsory lecture was done, they bowled me over to be first on the terminals. To be active, not reflective, and I am afraid, to *do* rather than *think*. Shouldn't these teachers be more concerned about kids' education than their training?

In a similar view, Sesame Street was designed to hook disadvantaged kids and those who had no tradition of studying. It showed them that learning can be fun. It hooked lots more kids and blurred the distinction between learning and fun; teaching them, by inference, that if it isn't fun, don't bother learning it. Not so. Education can be drudgery. Tedious hours of studying the classics. Is this why today's Johnnie can't read?

What is professionalism? The answer, as Tevya said,[5] is very simple. I don't know. But the main components are: defined body of knowledge of high intellectual content, defined standards of competence, examinations, code of ethics, and disciplinary capability.[6] These are present to a greater or lesser extent in the field of information systems. But to *be* a professional, you must first *consider* yourself a professional.

Then what is professional development? It is the continuing technical training necessary to attain or maintain professionalism. At present, podiatrists in New York State are required by law to take 35 hours of training and be recertified bienially. There is quite a stir among the legal profession in the same regard. The state is now considering[7] whether to require periodic refresher courses in four profes-

sional fields: those of physician, dentist, and optometrist, in addition to podiatrist.

I do not believe that training in managerial skills falls within the boundaries of professional development. The former is very important for those in management or aspiring to it but it is a different sort of problem and will not be addressed here.

OPPORTUNITIES FOR PROFESSIONAL DEVELOPMENT

So much for defining terms. Now to investigate opportunities for professional development. Let us consider degree programs, employer-provided training, and self-development efforts.

Degree programs

Technical degree programs may be considered the highest form of professional development. On the other hand, graduate business programs are aimed more at personal or managerial development, and are not within the framework of this discussion as defined earlier.

In passing, it is interesting to note the proliferation of new degree programs in Information Systems. Nunamaker[8] found in a recent survey that 77 institutions are offering a bachelor's or master's degree in IS, MIS, MS, or some variant. Apparently universities are finding wisdom in the excellent report by Bob Ashenhurst.[9] The content of these programs is positioned between business and computer science, and in my opinion, within five years will surpass both the MBA and MCS in importance. One department head I lunched with recently is so besieged by inquiries from industry for his graduates that he is considering retaining a telephone answering service.

Degree programs in computer science provide a lot of theory and analysis, and a little skills training. They are at the leading edge of such disciplines such as systems software for minicomputers, but they lag the user community in others, for example manipulating large data files. Thus an evening degree program in Computer Science may be undertaken as professional development by an applications programmer who aspires to work in systems software, rather than in systems analysis or management.

University Computer Science programs, while theoretical, provide the student the basis for understanding (and rendering value judgments) on technological advances. So a few evening university courses from the Computer Science curriculum may be a useful means of professional development even for someone not wishing to pursue the degree. Many colleges and universities offer Computer Science extension courses in non-campus locations for the convenience of computing professionals who are employed full time. As just one example, the New School for Social Research in New York offers a wide selection of evening certificate courses in computer science with a practical orientation—such as OS Debugging and Advanced Systems Analysis.

Most employers pay or reimburse tuition for job-related courses if good grades are attained. Bell Laboratories and other enlightened companies, as well as universities, may grant sabbaticals to staff members to pursue advanced degrees or research projects. There are a wide variety of computer-related professional development programs available thru colleges. The student must anticipate a major investment of time to attend class and to study—time away from home and family—but as with a financial investment, the return is proportional to the risk.

Employer-provided training

The employer benefits in many ways from offering good technical training to its computing staff. Obviously, technically proficient people work smarter instead of just harder. But also, turnover is reduced because professionals want to stay with an organization that offers technical growth. Recruiting is easier for the same reason.

The mere availability of in-house training facilities is not enough. It remains for each individual to take advantage of the opportunities afforded him. Indeed, the company should adopt an active rather than passive role by scheduling and monitoring attendance.

Technical training takes many forms depending upon size of company, rate of change in its computer installation, and degree of enlightenment. In a very large company or one with rapid growth or high turnover, classroom training is the mainstay. In a medium-sized or more stable installation, self-study courses on videotape are appropriate. And smaller or less advanced shops may rely strictly on sending people to external courses when and where they are offered. There is a place for all these modes of professional development in the typical company. If yours does not offer formal training, lobby for its adoption. Here are basic guidelines for an internal Professional Development program.

Establishing in-house training

It is best that job descriptions and responsibilities be formalized. If written job descriptions do not exist it could take many months until they are constructed to everyone's satisfaction. In any event, knowledge requirements must be defined for each position whether from job descriptions or from hearsay. "Before you prepare instruction, before you choose material, machine or method, it is important to be able to state clearly what your goals are."[10] Must Project Managers use Critical Path Scheduling Techniques? Are Senior Analysts responsible for discounted cash flow, or for statistical sampling methods? Do Lead Programmers prepare Warnier charts?

Next, sources for each level of knowledge are identified by reviewing multimedia courses or evaluating vendor courses. Alternatively, specific courses can be developed in-house—but there are three disadvantages to such courses. First, they are very expensive. For an experienced teacher to develop course material can require ten hours of prepa-

ration time for each hour of classroom time. For an inexperienced teacher working on new material the ratio can jump to 100 hours to one! Second is the effect of obsolescence. Consider a course in direct-access file design. For the course to be useful for programmers and technical analysts, it should be quite detailed and relevant to the facilities available. Student exercises should be directed toward the present installation. But when the direct-access devices or the installation access methods change, such training is instantly rendered out of date. The paradox is—the more specific the in-house course, the shorter its useful life. And a third disadvantage is the difficulty of assembling enough students and a qualified instructor in one place at one time.

Whatever the source of recommended courses, each job should have a list of recommended knowledge associated with it. Then present skills are rated and inventoried. Each member of the professional staff is asked to judge his proficiency in the technical skills important to this job—for training purposes only. New hires are asked to rate themselves similarly. Now, with skills required and skills available known, the training required by each individual can be planned in a cooperative effort between staff member and his supervisor. Another valuable resource in such a meeting is the systems project plan, which identifies the assignments each person will work on. These meetings to plan training should be held at least once a year. It is during such a meeting that the staff member should communicate his own goals for specific technical training. And it is very important that training be scheduled for specific times and should be considered just as important as project work. Training must not be allowed to slip because of problems on another assignment. Progress and performance in training must be measured and reported to management. It is not fill-in work and is not so treated by far-sighted organizations.

As stated previously, about five percent of available time should be spent in training. And about two percent of the department's salary budget should be allocated to out-of-pocket training costs. This sum will typically cover the cost of purchasing one or two multi-media courses as well as sending a number of people to external courses. Of course, the larger part of in-house training should use existing classroom or video courses at no out-of-pocket cost. The organization that is just launching in-house training should consider a rental agreement rather than purchasing courses outright. A more elaborate discussion of the entire in-house training process is available[11] and will not be repeated here.

Employer-provided external training

External training takes many forms. Obviously each vendor offers courses in use of its hardware and software. Unfortunately, some of the mini-computer vendors are suffering from growing pains as their marketing success surpasses their training capability.

Many private training firms offer good courses in systems methodology. Brandon Systems Institute, Ware Associates of Hudson Massachusetts, and Control Data Corporation's Institute for Advanced Technology come to mind.

One may measure the state of the economy by counting the number of such brochures that fill the in-basket. In lean times, external training is the first budget item cut by most user companies, so the provider companies go into suspended animation.

Even employers who make extensive use of in-house training facilities will continue to send selected people to external seminars. There is value in having Project Managers, for example, meet and socialize with their peers from other companies. They may pick up much of value from the successes and failures of other companies. And they may find out that things aren't so bad in their own shop after all.

The alert installation will collect, study, and file, written evaluations of all external (and internal) courses to be used in future planning. And the progressive installation will write the training firm to convey negative or positive views by attendees. This is appreciated by the vendor and helps to improve the breed.

Another form of external training that must be mentioned to this audience is the Professional Development Seminars offered by ACM, IEEE, DPMA, ASM, and other computer-related professional societies. Such seminars are offered both at the national and local chapter level. They are usually aimed more at benefiting members than at making money, and are usually less expensive and may be less formal than for-profit courses conducted by nationwide firms. Earlier in this paper, an annual training plan was discussed. Professional Development Seminars are rarely announced that far in advance. So the burden is on you to bring to the attention of your management any external PD seminar that supports your personal professional development plan. If no funds are available to pay the fee, ask for time off and pay it yourself. It will probably be tax-deductible. Under a liberalization of regulations, costs of training required to maintain or improve skills on the present job are a deductible expense (but training for a new job is not).

SELF-DEVELOPMENT EFFORTS

We have touched on the professional development offerings of universities, and have seen what employers can and should provide. Now let us consider what the individual can do for himself.

Establish a self-development plan

Consider your circumstances to the following questions:

1. Training taken in the last year
2. Books read in the last year
3. Current periodicals I read
4. Tasks that I perform on my present job
5. Personal qualities important on my job
6. Changes that I can see in the future
7. My strengths and weaknesses
8. Goals within one year
9. Goals beyond one year
10. Specific plans to accomplish these goals

Such plans might resemble the following:

(a) Read Weinberg's *An Introduction to General Systems Thinking* before September.
(b) Subscribe to *Computerworld* and *EDP Analyzer;* cancel subscriptions to *Sports Illustrated* and *Gourmet.*
(c) Begin a word list of all unfamiliar terms and their meanings. Carry this list and refer to it.
(d) Discuss ideas found in technical publications with coworkers. Practice active listening by restating your associate's view, in your own words, to his satisfaction.
(e) Find a professional development seminar on distributed data base.
(f) Seek an opportunity to give a short guest lecture on minicomputer applications in process control.

You should monitor progress on this plan and renew it periodically, at least annually. It is up to you; no one can do it for you.

Sources of professional self-development

Learning preferences and abilities differ. In our profession are found scholarly people and those who have come "thru the ranks." We differ in age, in temperament, and in experience. Thus training methods should differ. Some students respond favorably to video courses while others find them boring and would prefer to learn by studying reference manuals. Sources of the latter are well-known, and fortunately the former have also been documented.[13] The IEEE offers a wide variety of self-development opportunities to members, although not all in computers. For example there are self-study courses in engineering, on such subjects as Field Effect Transistors, and Digital Instrumentation; self-study courses in management, for example How to Start Your Own Consulting Business; approximately 100 field courses (to be held by your company); correspondence courses; and speed reading.

Many people learn by doing. "What I hear, I forget; what I see, I remember; what I do, I know." How to better learn about microcomputers than to build one? An IMSAI 8080 1K kit currently sells for $650, a saving of $350 over the assembled price. An expensive and time-consuming lesson, but one guaranteed to produce expertise and build a foundation for the future. (The 8080 chip, by the way, came out in 1974 priced at $300 in OEM quantities—at this writing, Radio Shack sells it over the counter for $18!)

Microcomputers may very well provide the at-home professional development wave of the future. Computer-assisted instruction has not yet become cost-effective except in special situations. But the Video Disk technology has prospects of providing huge quantities of read-only software, including course material, to microcomputer users—eliminating the traditional CAI requirement for large CPU connect time. The ACM Professional Development Committee is watching developments carefully.

Establish a reading program

Beyond any questions, the computing literature is overwhelming. The *ACM Computing Reviews* subscribes to 240 periodicals! Only a small fraction of their articles, and only a limited number of textbooks can be reviewed by *Computing Reviews*. But their staff of editors and reviewers does an admirable job—and the publication can be of great benefit to the reader. Clip or copy reviews of importance to you. File them and track them down when you have time or when your need is more urgent. Become a source of resource material to your associates. How can anyone be considered a professional who does not stay conversant with the literature of the field?

Each of the professional computing societies has a lead journal, most also have many special interest journals. They are far too numerous to list. Everyone ought to pick a few favorites and set aside time to scan them, marking or clipping articles to be read. Mine are *Datamation, Computerworld, EDP Analyzer, ACM Computing Surveys, IEEE Computer,* the *IBM Systems Journal,* and *Scientific American.* The September, 1977, issue of *Scientific American,* devoted to microelectronics, was a revelation to me. If sufficiently enlarged and studied, the Boolean logic of SLT circuits can be seen!

The *Honeywell Journal,* edited by Bob Bemer, had a fine format. It included microfiche in the inside cover, was multilingual in part, was printed on fine paper of the European standard size, and published significant articles with high quality graphics. Unfortunately it perished in the economic crunch a few years ago.

Take a rapid reading course and practice what you learn. One MBA student, overwhelmed with reading assignments, sat under the clock in the library and turned a page every time the clock's minute hand clunked. Even if he had not quite finished both sheets, he had the sense of the material. And 60 pages an hour is not a bad pace. Technical material may require more time, particularly if you own your books and write comments in the margin. There are several book clubs for computing professionals.[12] Membership will force you to set a reading schedule. However, many people believe in selecting their own books based on published reviews and personal recommendations, rather than reading books selected by committee. Discuss your reading and your opinion of it with your friends.

Other individual approaches

Your attendance here suggests two things: that you are a member of one of the AFIPS constituent societies, and that you are interested in professional development. If so, attend your evening chapter meetings. Practice your active listening skills on the speaker. Take notes ("what I do, I remember") and ask questions. Use the coffee or cocktail session to learn more, not to tell how you blasted out of a sand trap.

It is difficult, but useful, to find out what your coffee partner knows that may be interesting to you. Recently I learned, from a stranger, that the IRS master file consists of

1300 reels of tape! Such hearsay may not be totally reliable, but as a computing professional you have learned to render value judgments on incoming data.

Keep an audio cassette player in your car. Listen to speeches[14] you were unable to attend. Note-taking is a little more difficult here.

CONCLUSION

This paper has presented a variety of approaches to professional development. They range from formal university programs, to the semi-formal company courses, and the informal, personal endeavors in technical training. It should be apparent that there is enough variety and quantity of options that no one need to suffer from the malaise of creeping technical obsolescence. But it is an individual decision to construct an action plan and to persevere. No one can do it for you. Good luck!

DISCLAIMER

The citations given in the text and reference list were selected by the author. No attempt has been made to exhaustively list all training vendors, publishers, etc. The reader may use his resources to locate others, or is invited to contact the author for further information.

REFERENCES

1. *New York Times,* November 4, 1973.
2. "Navigation Between the Planets," *Scientific American,* June 1976, p. 63.
3. "Here Comes the PET," *IEEE Computer,* October, 1977.
4. *Data Processing in 1980-1985,* T. A. Dolotta, et al, John Wiley and Sons, New York, 1976.
5. "Fiddler on the Roof," Sholem Aliechem.
6. *EDP Analyzer,* "Professionalism: Coming or Not?" March, 1976 (and "The Question of Professionalism," December, 1968), Canning Publications.
7. *New York Times,* November 16, 1977.
8. Nunamaker, J. F. and Ronald Harding, *Status Report on Information Systems, ACM Curriculum Committee on Computer Education for Management,* August, 1977 (unpublished).
9. *Curriculum Recommendations for Graduate Professional Programs in Information Systems,* R. L. Ashenhurst, Editor, *CACM,* May, 1972.
10. Mager, Robert F., *Preparing Instructional Objectives,* Fearon Publishers, Belmont, California, 1962.
11. Gluckson, Fred A. and Michael G. Flannery, "Development of Computing Professionals," *Proceedings of the Tenth Annual Computer Personnel Research Conference,* ACM, June 15-16, 1972.
12. The Library of Computer and Information Sciences, Riverside, New Jersey 08075.
13. *Tenth Annual Guide to Audio/Visual Instruction for Data Processing,* J. Daniel Couger, Editor, CYSYS, Box 7345, Colorado Springs 80933, May, 1977.
14. Convention Seminar Cassettes, 13356 Sherman Way, N. Hollywood, California 91605; On the Spot Duplicators, 8817 Shirley Avenue, Northridge, California 91324. Lists of NCC and other speeches are available.

Professionalism in data processing management

by DELBERT W. ATWOOD, JR.
Utah State Board of Education
Salt Lake City, Utah

INTRODUCTION

Where do you stand as a member of the information processing community when the discussion turns to defining data processing as a "profession?" (Webster says, "One that engages in a pursuit or activity professionally." Webster also says, "Characterized by or conforming to the technical or ethical standards of a profession.") Professionalism is, "The conduct, aims, or qualities that characterize or mark a profession or a professional person." Black's Law Dictionary defines profession as:

> A vocation, calling, occupation or employment involving labor, skill, education, special knowledge and compensation or profit, but the labor and skill involved is predominantly mental or intellectual, rather than physical or manual. The method or means pursued by persons of technical or scientific training.

Black includes the following clarifying paragraph for the definition of a profession.

> The term originally contemplated only theology, law and medicine, but as application of science and learning are extended to other departments of affairs, other vocations also receive the name, which implies professed attainments in a special knowledge as distinguished from mere skill.

Occupational sociology literature suggests that modern technologically oriented societies indicate that the distinction once made between "Professional" and "Non-Professional" is blurred or vague.

ATTRIBUTES OF A PROFESSIONAL

In my opinion, the attributes of a professional are many. The qualities exhibited by a data processing professional place him/her above the "run of the mill" person. The professional:

- is knowledgeable, yet not overbearing
- listens, but does not always comment
- offers advice, but does not dictate
- works with people
- constantly strives to maintain technical competency
- is respected by his/her actions
- has extensive training in a specialized discipline supported by a body or research and theory
- supports a "code of ethics" developed by an association of colleagues
- performs a unique and essential service recognized by the community
- exhibits strong personal responsibility for his/her judgments and actions.

Individuals who score high on a professionalism measure exhibit a strong desire for implementing professional values among their peers.

One of the largest obstacles of any attempt in defining professionalism in data processing management is the lack of definite criteria as to what makes up a "professional" computer person. Some of these criteria might be:

- Education
- Knowledge
- Ethics
- Communication Abilities
- Public Relations Abilities

As we look back on *Webster's* definitions and characterize them as to our own particular situation, can we say that we are truly a professional, or merely standing on the outside looking in. Do we belong just for the sake of belonging, or do we belong so that we can be recognized as an individual who is knowledgeable about the profession and can be helpful to our management and companies as a result.

Leonard I. Krauss, in his book, *Administering and Controlling the Company Data Processing Function,* has a section on professionalism in the field. He states, "There are a number of factors which are characteristic of a profession. Rather than undertake a debate on whether data processing is a profession, the time could be better spent by discussing professionalism as it pertains to data processing. The aims and standards of data processing have been the concern of several data processing organizations, and also the USA Standards Institute."[3]

HOW TO ACHIEVE PROFESSIONALISM

A manager achieves a professional status from the actions of his peers and those who work for him. He is respected for his decisions. He is highly regarded by those who work for him and also those for whom he works. When a man is looked to for advice and counsel, and has gained the respect of others regarding the many aspects of the business he is engaged in, he is on the way to achieving the professional qualities described above. An individual can enhance his professional appearance through his training, education, association with others, his actions on the job as well as his personal life, and the activities to help others.

We, as managers of the information services areas of our companies or shops, have an obligation to our company, our management, our fellow employees, our staff, and also to ourselves to help make people understand our methods and procedures of operation, to advise our management so that sound, honest decisions can be made regarding our realm of responsibility. We must educate ourselves to be knowledgeable in the area of our pursuit or activity. We must work together to disseminate our knowledge to fellow workers and management, to realize the maximum benefit from our education. We must be loyal to our companies or firms, respecting the trust vested in us by them. We must conduct ourselves in an ethical manner at all times regarding the information processing profession and to any professional organization to which we belong. Your attitudes and actions mold for you the things that you are and the way other people see you.[1]

Are you a professional? Do you belong to a professional organization? George R. Terry, in his book *Principles of Management*, states "Active membership in a truly professional group can assist management development through attendance at meetings, informal discussions with other members, and the reading of the association's journal or official publication. Some associations stress individual self-development; others promote workshops, seminars and study groups. These experiences help to promote the important "develop yourself philosophy." Generally, most companies or employers wish to have their employees the most knowledgeable individuals in regard to their expertise. In order to accomplish this, they send them to schools and allow them to belong to groups that enable them to meet, share ideas, experiences and needs. They, in most cases, pay for the memberships and expenses. Companies and employers provide these additional fringe benefits to encourage professional development. This professional development should enable them to do a job better, quicker, and generally at a lesser cost.

When you are knowledgeable about a subject, your specialty, and you discuss and share this knowledge with your management, your peers and others, you become a professional. This recognition comes from others and can be helped by your membership in a professional group and your participation in all of the activities the professional group can provide.

Professionalism is sometimes associated with an honor society, or a certification of one type or another. This could also be in the form of an examination to license, to allow the individual to practice their specialty. This is characterized by the state boards to allow an M.D. to practice; the Bar to allow an attorney to hang out his shingle; the C.P.A. examination to permit the accountant who passes it to certify that his examination of a set of books is appropriate according to standards and acceptable.

Data processing, through DPMA, has constructed the certificate in Data Processing (CDP) as a test of knowledge in management, systems quantitative and qualitative analysis, programming, and other areas. Many professional groups joined together to form the Institute for Certification of Computer Professionals (ICCP). These groups recognized the need for a professional certificate to assist the individual in his personal development, recognition by his peers, his company or employer, and his development as a professional.

Many of the organizations have by-laws, codes of ethics, guidelines for professional conduct in information processing, and other written vehicles for the individual to follow to assist him in becoming a professional. Adherence to these precepts certainly would help us to be recognized as following a professional path.

Where do you stand in your own professional development and standing in your company, and within the organization to which you belong?

Are you going to classes to get your degree? Are you getting an advanced degree? Are you attending seminars to further your knowledge in your specialty subject matter? In our ever expanding field where specialized knowledge is mandatory, those individuals who demonstrate superior competence in the application of that specialized knowledge reap the rewards of progression.

The individuals who are looking for management recognition and who are looking for a greater professional posture in their work, may be selfishly motivated, but in the long run, they are on the road to becoming a professional. The knowledge and association developed in the process generally stimulates the individual to be more reflective about his work, and approach problems with a greater understanding and to bring to bear the new techniques and concepts learned to satisfy those problems.

Do you do your part? Are you becoming the professional your management would like you to become? Do you belong to a professional group, but never go to the meetings? You read they have excellent seminars and workshops, but when was the last time you attended one of them? Do you get a publication from the association? When was the last time you read it from cover to cover, or even selected articles? Can you quote from any of the articles about a particular subject matter that is coming up in a management committee meeting tomorrow? Can you refer to other publications, books, seminars, workshops, etc. to assist in helping to make management decisions?

Professionalism in data processing is beginning to emerge. It still may take a while to accomplish, but the tools are there, and if the right person or persons come along to guide the tools, the guidelines, ethics standards, methods of control and other details will be pulled together and be recog-

nized by all information processing personnel and organizations, and will be thus enforced by them or their peers who make up the organizations. The profession is growing by leaps and bounds, and some restraint or control is needed before it gets out of hand.

CONCLUSION

I believe that professionalism makes a difference, because an individual who is looked up to, respected by his associates and others, who exhibits the traits that place him a bit above, has a better chance for success and overall happiness in his life. A doctor, a lawyer, or an engineer work very hard to become educated and licensed to practice their specialty; likewise, data processing managers work very hard to become educated and knowledgeable in their specialty. Licensing is a way off, but maybe it will become a necessity before professionalism is a reality, certification is here now and is a step toward achieving the recognition and respect of our employers, our employees, and our peers.

BIBLIOGRAPHY

1. Atwood, Delbert Jr., Editorial, *Data Management,* June 1977.
2. Black, Henry Campbell, *Black's Law Dictionary,* St. Paul: West Publishing, 1951.
3. Krauss, Leonard I., *Administrating and Controlling the Company Data Processing Function,* 1975, Seventh Printing, Prentice Hall, pp 156.
4. Terry, George R., *Principles of Management,* 1972, Sixth Edition, pp 516.
5. Weinthal, Donald S. and Garrett J. O'Keefe, Jr., *Professional Orientation of Broadcast Newsmen,* August 1973.

Professionalism—A question of semantics

by EUGENE B. SMITH
U.S. Department of Agriculture
Beltsville, Maryland

INTRODUCTION

The topic of professionalism is of crucial importance to the ever-increasing group of individuals dedicated to the design, production, dissemination of knowledge, and application of the computer. There is a strong desire for professional recognition by large portions of this group. This paper focuses on the need for an appropriate definition for a computer professional. Questions are raised concerning the various requirements and related responsibilities which should be associated with recognition as a professional. The element of controversy over numerous issues will serve to clarify, if not resolve, the parameters which outline the future data processing professional.

PROFESSIONALISM DEFINED

In order to establish a baseline from which this dissertation can proceed, there is some value in presenting a "Webster's" definition of terms which are of fundamental importance to the topic. These terms include the following:

- Profess—To practice or claim to be versed in (a calling or profession).
- Profession—A calling requiring specialized knowledge and often long and intensive academic preparation.
- Professional—Characterized by or conforming to the technical or ethical standards of a profession. (Participating for gain or livelihood in an activity or field of endeavor often engaged in by amateurs).
- Professionalism—The conduct, aims, or qualities that characterize or mark a profession or a professional person.

The term professionalism has different meanings to different people. Ralston[1] defined professionalism by saying that ". . . a professional is someone dedicated to rigid training rather than immediate self-gratification. As such, "professional" bespeaks not elitism but a state of mind."

In an attempt to define the term professional by example, it would seem that there are at least two distinguishable groups. The first group might be called the "learned" professional and it might be categorized as having demonstrated a profound knowledge or scholarship. This group includes the universally recognized titles of doctor, lawyer, and engineer. Their professional recognition is based on an attained level of knowledge in a specialized field; a certification that they have reached a specified level of knowledge; and legal registration which gives them the right to practice their profession within certain constraints.

A second group of professionals may be categorized as the "other" professionals. These individuals may be identified as being associated with the second definition for professional which was given previously. That is, we may consider that they participate for gain or livelihood in an activity or field of endeavor often engaged in by amateurs. This group would include the professional bowler, the professional boxer, the professional football player, the professional golfer, and others. In addition to professional athletes, we might also include the professional politician, the journeyman electrician, and the journeyman plumber. The professional athlete has recognition based on his level of ability in his chosen field. In most cases this status is governed by membership in a professional association. The professional trades normally require an apprenticeship associated with a trade union. Advancement to the journeyman level implies the attainment of a reasonable skill. In some cases, local governments require that a licensed journeyman be responsible for the quality of work performed.

It may be helpful to associate the "learned" and "other" professionals on a continuum. In terms of respect or service to society, we may place the "doctor" at the high end of the continuum. In terms of income or monetary reward, we would have to place the boxer at the first of a continuum. In the case of either the "other" professional or the "learned" professional it is possible to identify the credentials associated with their professional status. In the case of the computer professional, it is important to identify the credentials associated with the achievement of a professional status. In addition to other things, these credentials are meant to insure a minimum level of competency and adherence to certain standard practices. If suitable credentials or criteria do not exist, it is important that these be identified or established to the satisfaction of the individual, his peer group, his employer, and society at large.

ELEMENTS OF DATA PROCESSING PROFESSIONALISM

In addition to identifying the various factors associated with the achievement of a high level of professional recognition in the field of data processing, it is important to pursue the parameters or limits associated with each factor. It must be possible for individuals to work toward achieving the status of a professional and know when they have obtained it. Such recognition must extend beyond the peer group to the employer and to society as a whole. Once a high level of professional recognition is possible, it may be that many individuals in data processing may not be suitably motivated to try to achieve it.

Perhaps the first question to be resolved is why we wish to achieve a professional status. What is our purpose? Is it to satisfy some personal ego requirement which would allow one to be a member of an elite group? Do we wish to improve our monetary status within our group? Do we wish to improve the moral and ethical level of our peer group? Are we interested in improving our level of service to both our employer and society as a whole? We are associated with the use of a technology which can have awesome implications to all of mankind. It is important to give consideration to the underlying motivation for our desire for professional status.

The term profession implies the existence of a certain level of specialized knowledge. One might question the make-up of this body of knowledge and how it can be validated. The Institute for Certification of Computer Professionals[2] identifies the content of the Certificate in Data Processing (CDP) examination as containing sections on Data Processing Equipment, Computer Programming and Software, Principles of Management, Quantitative Methods, and Systems Analysis and Design. The CDP eligibility requirements consist of the equivalent of 60 months full time related experience. An academic alternative allows a substitution of designated academic work for up to two years experience. This sets an example for a test that, to some extent, validates a level of knowledge at one point in time. Is this "the" common body of knowledge which the computer professional should have? Are there several specialties as there are in medicine or engineering?

If we could assume that this is "the" common body of knowledge that is desired and that the eligibility requirements are suitable, there are still many other factors which must be considered. Is there a need for periodic re-certification? Both DPMA and ACM have Self-Assessment programs under way which should help an individual identify his weak points and provide references for correcting them. Continuing education in many forms is available on almost any topic. The mechanism for maintaining a current knowledge of the state-of-the-art is present but the verification vehicle is missing.

Consider the ethical implications of professional status and how they might apply to this situation. In terms of loyalty, Pletta and Gray[3] point out that ". . . the professional's primary loyalty may be to: (1) society; (2) his peer group; (3) employer or client; or (4) himself. Only if he is free to elect the first of these does society confer upon his peer group the status of a profession." Loyalty will greatly influence a course of action when a question of conflict of interest arises. Questions involving privacy and security are also related to ethics.

While each major professional group has a code of ethics associated with membership, one might question the adequacy of the requirements. It would seem that the enforcement provisions and mechanisms leave something to be desired. Even the old line professional groups such as the doctors and lawyers have great difficulty in adequately enforcing their respective codes.

The legal implications of professional status are primarily related to a licensing process. Municipal, state, and federal agencies often have a role in licensing. The various legislative groups must enact specific laws to create the license mechanism. Lord[4] goes to great lengths to distinguish between the licensing and certification processes. The licensing process must also have a sound basis for enforcement.

It is appropriate to identify the various players and their role in the question of professionalism. The list includes the following:

Academia—The majority of the formal training will be done in the numerous colleges and universities. These institutions have a responsibility to provide training in an accepted common body of data processing knowledge. The curricula must be kept current with the changing technology. In addition to a well rounded education they must attempt to instill a sense of responsibility, morality, ethics, and general professional awareness in the students.

Professional Groups—The role of continuing education carried out through professional meetings, quality publications and innovations such as the self-assessment programs is a primary responsibility of these organizations. Their role in certification, and/or some accepted vehicle for determining the minimum level of knowledge is crucial. An organization such as the ICCP, which receives wide support from most related professional groups, is needed to serve as the testing and enforcement agency. The professional group must strive to represent and provide a unified voice for the membership.

Employers—If a significant professional status is to be achieved, it must be recognized and supported by business and industry. This recognition could take the form of consideration in the hiring and promotion processes.

Governmental Agencies—In the event that some form of licensing or registration is appropriate, the government agencies would be expected to establish the requirements, procedures, and control mechanisms for enactment of legislation.

Society—If society does recognize a professional group through general acceptance and the enactment of legislation, it has the right to expect a high level of moral and ethical responsibility. Society also has the responsibility to speak out when the actions of the professional adversely affect segments of the public.

The Individual—When an individual achieves the status of a professional, he must accept certain responsibilities along with the related benefits. Some of these responsibilities are legal and some are social. In the event that he fails to meet the responsibilities, his peers must be willing to take appropriate corrective action.

Each player has a role and associated responsibilities in progress toward professional recognition for the Data Processor. The failure of any player to assume his proper role and responsibility will impede the progress.

SUMMARY

Professional status will be achieved in the data processing area. The speed and degree to which this is accomplished will depend on the dedication of the individual and his level of support for the data processing professional societies and related groups.

REFERENCES

1. Ralston, A., "The Roles of ACM IV. ACM and Its Members: Relevancy and Responsibility, ACM President's Letter," *Communications of the ACM,* Vol. 16, No. 7, July 1973, pp. 397-398.
2. Certificate in Data Processing Examination, *1978 Announcement,* Institute for Certification of Computer Professionals, N.Y., 1977.
3. Pletta, Dan H. and George A. Gray, "Engineering Professionalism—A Light at the End of the Tunnel," *Engineering Issues,* ASCE, April 1977, pp. 157-163.
4. Lord, Kenniston W., Jr., "Licensing, Certification Not Same Process," *Computer World,* October 24, 1977, p. 23.
5. Finerman, Aaron, "Professionalism in the Computer Field," *Communications of the ACM,* Vol. 18, No. 1, January 1975, pp. 4-9.

Continuing education opportunities—A mark of a profession

by ROLAND D. SPANIOL
Eastern Illinois University
Charleston, Illinois

INTRODUCTION

Authorities indicate that one of the marks of a profession is the availability of continuing education to its members. Education that will tend to keep the members current with their technology and skilled in applying it in their work. It seems reasonable this should be a requirement of information systems if it is to be designated a profession.

AN INTERPRETATION OF PRESENT CONDITIONS

Educational products for people in information systems occupations come from a variety of sources including hardware manufacturers, software houses, government regulated industries, universities and colleges, profit motivated schools, businesses, etc. Based on the large number of advertisements in the mail each day, it is easy to identify the variety of educational offerings that are available. There are many; they are offered in many big cities; and they cover a wide range of topics.

The present situation with respect to educational offerings is summarized in the following paragraphs.

The enormous number of group and self-instructional courses presently on the market cover the majority of the subject areas revealed in the DPMA Education Foundation's "Survey of Educational Needs and Desires."[1] For example, 25 of the 34 most frequently mentioned subject areas in the survey are covered by one or more current promotional brochures.

Is the demand for these courses so great that the needs of the information systems practitioners cannot be met? Is the demand being driven by a desire on the part of the information systems practitioners to be professional? Do you think the courses would be as heavily advertised as they are if this were the case? In short, most of what the information systems practitioner says he needs is probably available; but to be blunt, much of it is probably the "paper tiger" variety.

At the present time prospective registrants have no feasible way of evaluating the quality of an offering beyond the general reputation of the firm that markets it or the personal recommendation of past participants with whom they may have had some chance contact.

Many private educational offerings are priced well beyond the reach of those who probably need them the most. This unfortunate situation is due, of course, to the high cost of producing and delivering an educational product in combination with too few buyers spread over too many sellers.

To summarize present conditions, it can be said that most of what information processing professionals say they need is available. It may not be in the precise form they would like it to be, but it is available. Their problem is to find it among the plethora of offerings while avoiding poor quality and figuring out a way to pay for it.

If it is true that the information systems practitioner's educational needs can generally be met with current offerings in a variety of big cities, but that the cost is high, what is the solution? The one obvious solution that comes to mind is to cut the number of offerings, teach to a larger group at a time, and lower the tuition for each student. Those of you involved in higher education in recent years will note this is not an original solution. Those of you involved in offering the courses from the private sector may be curious how this can be accomplished.

EVALUATION TECHNIQUES

What I am suggesting is not new either, it is supply and demand aided with market analysis, evaluation of the offering, publicity, and a booking service.

Market response to an educational offering is a good indicator of the worth of the offering. If the course is popular and makes money, it can be repeated; when it loses its appeal, enrollments decline, and profit wanes, it will be dropped. This is not an unknown phenomenon (except maybe in public education). As a matter of fact, for educational offerings from the private sector it is principally the law of supply and demand that decides whether a course is to be repeated. Demand and enrollment are the required measures of the quality of the offering.

Reaction to an educational offering is a commonly used indicator of the quality of the offering. The technique is familiar to you; the attendees each receive an evaluation form near the end of a course and the attendees rate the presentor, the presentation, the content, the facilities, etc.

Since it is easy to do and results are quickly available, it is often used with market response as the way to improve the offering, to determine whether or not it should be offered, which instructor is getting the best evaluation, type of facilities most desired, etc. Later, you will see that it leaves room for improvement as an evaluation instrument, but it is an important measure of how well the program was accepted.

Learning that has taken place because of an educational experience is a better measure of an educational offering. For this situation, suffice it to say that learning is the ability to recall a concept, knowledge, technique, or item of information in a testing situation. Comparing reaction to learning, it can be seen that learning is a better measure of the success of a program than reaction because with learning we know that the program message is getting through to the enrollees.

Learning is more difficult to measure because if we are to be precise, we must test before we teach and then test after we teach so we can develop a statistical correlation and/or confidence level. Because it is more difficult may be the reason we rarely see testing done in courses offered by the private sector. What do you think?

Behavior changes in an individual that are perceptible to others as the result of an educational experience is an excellent measure of the value of that experience. But, measuring behavior changes is much more complex, difficult, and costly than measuring learning or reaction. It usually requires experts in testing, is usually administered to a special group or groups at a time, is usually done on contract, etc. Though difficult, measuring behavior changes is certainly a valid evaluation process and should be considered in special situations.

Results is what everyone wants. Why not measure the contribution of educational programs based on whether the enrollees get satisfactory results from implementing their new found knowledges, techniques, skills, etc.? Probably because results are more difficult to measure than market response, reaction, behavior, and learning combined. Not only that, but it is difficult to be sure that the results achieved are related to the instruction. Much more work needs to be done relative to measuring results.

IMPLEMENTATION OF A NATIONWIDE PROGRAM

Catalog of programs

Initial efforts should be directed at facing the formidable tasks of identifying programs and materials currently in existence that will meet the members professed educational needs. Indeed, the greatest contribution may be in seeing that an ongoing cataloging process is set in motion to identify to the membership where and when they can enroll in programs they feel they require to keep them abreast of current technology. A catalog of these program offerings could be made available on a subscription basis to libraries, businesses, individuals, etc. In order to be kept current it would have to be updated no less than monthly. The listings should be complete with course outline, presentor, dates, places, costs, etc.

Booking service

A natural byproduct of a catalog of programs is a nationwide booking service, a centralized office where persons could enroll in the courses of their choice. This office could be the central point for all tuition, enrollments, evaluation activities, etc.

EVALUATION PLAN

Evaluation of the programs cooperating in the project would be a major activity. It is recommended that reaction evaluation be the first method used. This method should be used for all educational offerings made a part of this service. The courses in the top 10 to 25 percent should be given special publicity such that they would be widely recognized as popular courses.

As the project matures, the testing for learning process should be initiated. Attempts should be made to see if learning does take place. In the programs where learning is greatest, the courses should be given special publicity so they would become widely known as good programs to take to learn the needed information.

In some special cases, attempts should be made to determine changes in behavior. Because this evaluation method would be difficult to implement and would have marginal influence on this project, it will not be dealt with in this paper.

Results evaluation is an interesting method and in a broad sense could make some real contributions on a nationwide basis. Initially, or in the foreseeable future, there does not seem to be much results evaluation can do for this project and will not be dealt with in this paper.

On a nationwide scale, however, it would be interesting to see if the information systems professionals could identify the knowledges of the citizens about an issue such as privacy. Then, after implementing a large scale public education program on the issues of privacy, follow this up with a post test. Enough said.

WHO COULD MANAGE THIS CONTINUING EDUCATION PROJECT?

Possibly the DPMA Education Foundation. Or maybe any of the professional organizations. How about AFIPS? Perhaps because of my prejudices, I think the DPMA-EF is best suited for the job. They are an IRS 501 (c) (3) not-for-profit corporation. They have no members, and they were organized with the major objective of providing education to the information systems practitioners. Why not let them know how you feel about having them sponsor such a project.

WHAT WOULD MOTIVATE COOPERATION WITH THIS PROJECT?

Have you ever sat through a two or three-day course or a semester one for that matter and felt you were wasting your time? Do you think that experience was unusual? Would you like to do something about it? Of course, we have all had bad experiences on occasion. And, if there was a formal process where we could input our thoughts about a course, we would have done so. The primary motivation for an individual to cooperate in the evaluation process is improved education. The more information systems practitioners we can get to evaluate the course offerings they take, the better chance we have to weed out the less effective courses.

Getting cooperation from the presentors may initially be more difficult than getting cooperation from the individuals taking the instruction. After all, what is in it for the presentor? Of course, it depends upon the management of the project, but I would hope the presentors would soon see the merits and become the proponents of the process.

The project incentives which will engender presentor participation will be positive incentives. If the project is to be successful, the presentors must want to participate, must strive for the opportunity to be listed in the catalog, must encourage enrollees to participate in the evaluation, etc. What will encourage this eagerness?

The possibles include the following:

An honor roll of blue ribbon presentations. A panel of judges selected by the participating presentor organizations would select a limited number of offerings for special recognition. The top two or three presentors would be selected from this group for a special award or prize.

The opportunity of participating in a cooperative nationwide publicity activity. Through advertising in the trade press and other publicity, the offerings of the cooperating presentors would be made available to the information systems practitioners.

A catalog of the offerings of all the presentors. The listings would be by presentor organization, by subject, by city, and by date, for example. This catalog would be made available on a subscription basis. The subscription price would be low enough to be attractive to public libraries, companies, and certain other training organizations, and governmental agencies.

The opportunity of participating with experts in the process of developing evaluation techniques. It is anticipated that the number of presentors involved would be great enough that they could employ testing specialists to work with them on the evaluation processes.

Altruistic as it may sound, I hope some of the presentors would want to participate for the simple reason that they would like to improve their offering. Just because they want to do a better job.

Then, there are other kinds of reasons. Some information systems practitioners would like to show that they have the trappings of a professional. These might include a college degree, a certificate such as the CDP, and some would even prefer a license to practice.

In any event, it seems reasonable that if a profession offers continuing education opportunities to its members, its members may want to have a formal method of showing they are taking advantage of this education.

There is such a system where courses are evaluated and are then assigned whole or fractional units of instruction. These units are called Continuing Education Units and are known as CEU's. It seems reasonable the programs cooperating with this project could be dealt with in a similar fashion and the record keeping for the participants could be kept as part of the project.

SUMMARY

In this paper the surface of an idea has been presented. What we need now is someone who wants to develop this idea into a reality. I hope you will agree that there are several prevailing reasons for our industry to formalize and strive to improve the continuing education aspects of the industry. We in the industry will be the last to know we are in a profession or when we became a profession. Let's make sure we have taken every opportunity to be as professional as we can be now that we are a part of that group.

REFERENCE

1. Spaniol, Roland D. and Eugene Smith, "The DP Professional and Education: Preliminary Survey Results," *Data Management*, August 1976, pp. 30-31.

So you think you are a professional?

by G. GARY CASPER
Weber State College
Ogden, Utah

INTRODUCTION

The human ego provides continuous opportunities to explore the richness and variety of human perception. It is capable of excruciating truth and insight as well as delusions of grandeur that obscure truth and reality. The maturity of an individual can be defined in relation to his ability to differentiate between these extremes.

The computing practitioner, being human, is no exception. His existence in a rampant technology substantially lacking established codes of standards, conduct, and practice accentuates his capability to delude himself and makes more difficult a clear understanding of the reality of his experience in a complex environment. He can to a large degree establish his own criteria of accomplishment, yet must feel some concern of exposure as he realizes the effects of the rapid change forced upon him by his technology.

This paper postulates two resulting classes of behavior observed in the industry, one which could provide considerable benefit to the individual in coping with his anxieties while the other appears to be counter productive.

The first category can be described as behavior reflecting a desire to be identified as a "professional" and as such be associated with practitioners characterized as possessing competence, integrity, contribution, and esteem.

The second category represents a potpourri of behaviors generally designed to avoid exposure of the anxieties experienced from the self-doubt engendered among those responsible for the implementation and control of rampant technological change.

The purpose of this paper is to investigate the current level of maturity in the industry which has been defined as differentiating between reality and protective delusions, and to make recommendations upon the conclusions drawn.

One cannot but be impressed with the frequent and pervasive use of the objective "professional" in reference to the many authors and speakers represented in the trade press and at various computing conferences. It appears that the proverbial "Everyman" considers himself a professional although considerable debate exists as to exactly what is the meaning of the term. In fact, one might express curiosity at the apparent egotism evidenced in the employment of the term. While clearly proposing their own professional status, most express serious doubts about the credentials of any others. What then are the essential criteria of a professional?

To begin, professionalism in the computing industry cannot be understood unless one first clearly understands what a profession is generally and the environment in which the professional finds himself.

The age in which we live has been characterized as one suffering from "Future Shock," a term defined by author Alvin Toffler as the effects of ". . . too much change too fast." As our world, environment, institutions, our very lives and personal values reel under the onslaught of pervasive change, we respond in dazed, often ill-defined, and inadequate ways.[1]

As practitioners of the computing arts, we are thrust into the forefront of this technological revolution facing the constant realization of technological obsolescence. Ours is a young industry with few if any age-old standards or solutions to an ever increasing complexity of opportunities and challenges. How are we to survive for a career that may span fully four decades?

Certainly education is offered as a possible solution, but who is to prepare and disseminate appropriate state of the art courses for an increasingly diverse and specialized body of knowledge? Are we not facing a dilemma of enormous magnitude whose solution will require the resources and commitment of significant numbers of us? What will be the cohesive force to organize and coordinate our efforts?

Has not our answer been to some extent to join together with peers of common interests to develop programs, materials, and vehicles with which to attack such problems? In fact, are we in this regard much different from other industries which have experienced similar needs? I believe not, and anticipate the increasing need for a developing, responsible, and viable profession.

What exactly is a profession, or more specifically, a learned profession? To understand the concept, one can look to other so called professions for criteria. One of the more common is tradition. For example, the story is told of a doctor, engineer, and politician arguing about whose profession was the oldest. The doctor said there was little doubt that medicine was the oldest profession because humanity was preceded by the first medical operation—the removal of Adam's rib. However, the engineer objected arguing that engineering preceded medicine by virtue of the

great project of creating the world from chaos. To this the politician inquired, "And who do you think created the chaos in the first place?"[9]

While data processors may not have a claim to old age for their profession, it might be observed that many feel we certainly have a legitimate claim for creating chaos.

Perhaps some light might be shed on this matter by looking at the definitions of a profession and a professional.

The simplest definition Webster offers is that a professional is one "following an occupation as a means of livelihood or for gain." Such a definition qualifies virtually anyone and everyone. Somehow, though, the lack of distinguishing criteria diminishes the concept and leaves it less than useful.

Hardly satisfied, definition two catches the eye. Professional is . . . "pertaining or appropriate to a profession." Further investigation reveals that a profession involves . . . "a vocation requiring *knowledge* of some department of *learning* or science." Here at least are some minimal criteria beyond merely holding a position in the industry (i.e., knowledge and learning). A professional is one "engaged in one of the *learned* professions," and is also one . . . "RESPECTED BY THE PROFESSION."

In this more lofty view of professionalism, one discerns other criteria pertaining to . . . "professional character, spirit, or methods," and again, the . . . "standing practice, or methods of a professional as distinguished from an amateur."[10]

Clearly there is a world of difference between the perceptions of a professional conveyed by these later definitions from the extreme of mere employment.

One might conclude then that to be professional is merely a matter of definition and clearly that a definition could be devised to cover anyone desired. However, to do so seems hardly productive as it serves only to camouflage any real differences among practitioners. What is sad is that such a strategy appears to be effectively implemented in the computing industry. At best passive, yet more often active resistance to any attempt to differentiate significant variables among computer practitioners is prevalent in the industry. One must ask why?

A complete answer would have to consider many possibilities among which would be the desire to avoid exposure of the incompetent, insecure, paranoid, and even the amateur. However evident such motivations and the real need to address the problems of such people, the intent here is to investigate the competent practitioner to attempt to understand his apathy in such matters and his reluctance to identify the characteristics of his own competence. Such serious deficiencies betray an absence of maturity among such practitioners.

Maturity is a state of being; an expression of the full development of an individual's ability to differentiate reality and delusion. If one can demonstrate a serious deficiency in such development of a person or groups, then one has the basis to challenge its maturity.

This is precisely what is proposed here. Since the computing industry is seriously neglecting the critical need to differentiate professional development among its numbers in that it tolerates a simplistic blanket usage of the term professional that masks the incompetent and amateur in its midst, then in that respect it lacks maturity. It is guilty of ignoring the reality of differentiating criteria among practitioners and of deluding itself into believing no such discrimination is necessary.

Not all practitioners of the computing arts are guilty of such lack of maturity. Many have addressed the problems, have worked diligently to devise sound definitions, and have tried to accomplish the identification of professional characteristics, but unfortunately many more have chosen to negate or compromise any achievements obtained. For the most part such contributors have been ostracized for their efforts and bullied into silence by the masses whose comfortable but erroneous self-image would be shattered by a glimpse of the truth.

Frankly, there is no such thing as a data processing professional! At least, not in the sense most people in this industry seem to use the term. We have no clear example of a professional directly comparable to a doctor, lawyer, or CPA. It is suggested that the term is being misused and in reality twisted to meet the ego needs of an immature industry.

However, the potential of a profession exists and could easily be established through the efforts of significant support from practitioners. What then would be required to qualify as a profession?

Richard Canning, in the EDP Analyzer (December 1968) listed a number of criteria by which any group could judge if it would qualify as a profession.[9]

1. A profession has a defined body of knowledge of high intellectual content acquired by training-in-depth.
2. A profession has defined standards of competence and certification that the professional meets those standards.
3. A profession has a code of ethical behavior.
4. A profession has at least one professional society aimed at advancing the welfare of its members.
5. A profession has responsibility to society to perform in a competent, ethical manner.
6. The members of a profession generally are not under the control of non-professional management (with regard to professional decisions).
7. The members of a profession have a loyalty to their profession which may transcend their loyalty to their employers.
8. The members of a profession may be licensed by the state to practice the profession if it affects such things as public health, public safety, property rights, or schooling of the young.
9. A profession has the right and ability to eject someone from the field for being incompetent or unethical.

Now the question is . . . Does the computer field qualify?

Certainly a body of knowledge in the computer field is being defined. Academic programs, educational programs of professional societies and an increasing body of literature are continually refining our understanding of minimal knowledge requirements. However, the technological and infor-

mation explosion will require diligent attention to the maintenance of these knowledge requirements. Who is to fund and man these efforts?

We have a certification program through the Institute of Certification of Computer Professionals (ICCP) which is being seriously developed, expanded, and implemented. But still there are few if any clearly defined standards of competence, although efforts are under way to attempt their definition. Who should be responsible for such standards definition?

The ICCP has a published code of ethics as do most of the existing professional societies. However, most of these codes are "Mother and Apple Pie" documents with no enforcement capability. Who must take on the continuing and sensitive task of defining ethical behavior, and more critically, who will be responsible for the enforcement of such codes? Only ICCP has an enforcement mechanism in place but it has yet to be exercised.

Most data processors are not independent decision-makers, nor has there been much indication of a willingness to transcend their loyalty to the company and their paycheck. Furthermore, many argue that one ought not take responsibility for questioning the propriety of their management or employers. What reinforcement should and could be provided to encourage responsible professional behavior in the public interest regardless of potential company reprisal?

Notice the need for the recognition of a legitimate and perhaps necessary licensing role. In spite of such recognition by increasing numbers of responsible leaders in the industry as well as the private and public sectors of our society, an overwhelming majority of practitioners now reject the concept.

Finally, no one to my knowledge has ever been ejected nor disciplined by any society for being incompetent or unethical.

What all this means is that the computing profession, if there is one, is only an emerging profession and much needs to be done if in fact it is ever to achieve a legitimate status.

Many practitioners argue and act as though such a situation were desirable and that there is no need for concern. With many of the benefits of a profession emerging anyway, why be concerned with the legitimate recognition of the profession? Let things take a slow, natural course and all will come out in the end. In the meantime no one need be threatened and all can use the term "professional" as they please at no cost to anyone.

Here again is a manifestation of immaturity, for the satisfaction of the ego need to be regarded as a professional is accomplished through the delusion that such a choice is within the power of the computing industry alone. Ignored is the reality that the public, legislatures, the courts, employers, and other professions are expressing an entirely different point of view. While naively awaiting recognition of professional status, events are occurring that may well forever deny such status to computing personnel.

In December of 1971, the U.S. Department of Labor's Wage and Hour Division decided that programmers were not exempt and should not be considered professionals.[7]

This precedent supported a January 1976 decision handed down by a U.S. District Court judge in Tennessee that again ruled programmers not exempt since they did not satisfy the provisions of the Fair Labor Standards Act of 1938, Section 213 which specify that "any employee employed in a bona fide executive, administrative, or professional capacity" is exempt from overtime pay.[1]

Seeking professional recognition, programmers have organized professional societies, sponsored professional activities, and encouraged professional development of the practitioner, but also all to no avail.

What happened? Where did they slip up? First of all, in their lack of attention to the necessary legal definitions and secondly, because of divergent motivations among their numbers.

Lacking a defined and recognized professional status sealed the fate of programmers who now face legal precedence in their struggle to achieve a profession. Are the rest of us likewise to be denied?

Close upon the heels of this bombshell has come the controversy in several states during 1976-77 concerning the tax status of computer software. The battle continues and the Data Processing Management Association (DPMA) as reflected in an August, 1977 position statement is arguing, ". . . whether data processors will receive equal treatment under tax laws as services by other professions."[8] This is an interesting argument considering the status of the so called computer profession, and it certainly puts the industry on notice to its exposure unless its professional status is established.

But the question now becomes, how long will this process take? A corollary is how much time is available for completion?

The answer depends upon projections of current and future events now evident and believed to be inevitable. Among these are governmental intervention such as has occurred with legislation involving privacy and electronic funds transfer systems.

Concerns that the public interest must be protected is causing legislatures to become increasingly sensitive to the impact of computer technology. How long are they going to tolerate the technology without regulation?

If one accepts the inevitability of such regulation within limited time frames, the argument becomes compelling for accelerating formation of the profession and regulation within it. In fact, the passage of regulatory legislation could make all the prior concerns empty rhetoric.[5]

It is in the belief that such would happen, and especially that it would come in a relatively short time frame, that has prompted many of the need to address the controversy. However, such legislation is not without peril and inadequate preparation must be avoided if possible.

Such has been the appeal of this author in the past who has been concerned that the industry is faced with potential for the establishment of a profession, and that it would seem prudent to prepare for it.[2–6] The argument has been made that certain things must be done to prepare. In particular, identification must be made of who would be a professional, what he would do that would differ from non-professionals, and how he would perform those activities.

Therefore, the real issue is the formation of a legitimate profession, regulated to protect the public interest, to insure the quality of the professional, and to provide him with the working tools with which he will perform.

If it maintains a head in the sand attitude, the industry is likely to find itself one day regulated by government established in desperation to control a technology the public perceives as much too powerful, self-serving, and threatening.

Such control may even exclude a DP profession and relegate it to the role of technicians employed by and regulated by some other profession such as accounting, or even worse, give rise to unionism. There are ample reasons to fear these eventualities.

If the computer industry is to participate in this regulation of computer related areas rather than perform as technicians under the guidance of other professions, then we must establish ourselves as acceptable professionals and in short order. Are we going to accept technician roles to other professionals? Or worse yet, unionization as technicians of the laboring class?

Let it be again perfectly clear that the use of the term professional in our industry is merely a dream or hope for the future. True, many are working for the reality but the vast majority are either apathetic or actively opposed to the concept. It is this apathy and opposition which must command our interest, for a lack of understanding of the motives of such opposition may well spell out doom to the efforts to legitimatize a computing profession. What then might we do to create a legitimate professional in our society?

One formal effort to achieve such professional status is the certification movement. Here is attempted a clear and comprehensive articulation of peer evaluation and approval with attendant subscription to a continuing practice of ethical behavior. This, of course, is the realm of the Institute for Certification of Computer Professionals (ICCP). This effort is essential to the development of a professional group of people and represents the most significant industry effort to improve and regulate itself. At this level one finds most of the generally mentioned and accepted attributes of the definition of a profession.

Culminating the work of many these past years, the certification movement promises the industry its best potential of achieving a profession and the practitioner his best chance to aspire to a truly mature professional status.

Full definition and development of a computing profession under the umbrella support of most of the significant computing societies is possible through the Institute for Certification of Computer Professionals (ICCP). Charged with the task of identifying significant discriminating characteristics among practitioners and certifying against such gives rise to hope for a profession. Definition of and attention to the characteristics of such a profession in addition to programs directed toward their achievement remain the best hope of the industry. However, while these efforts are often spoken of and are of interest, such is again not the purpose of this presentation. What is critical here, is the serious lack of support for these efforts.

The trade press ranks with superficial and often foolish criticisms of the serious attempts to improve the situation of computer professionals. Rarely, however, rises a thoughtful defender to spoil such attacks.

Rarely do we call our employers attention to these credentials or encourage our colleagues to acquire them. Rarely are such credentials rewarded. Rarely do we volunteer the time or effort to assist those making the sacrifices in our behalf. How then can we truly consider ourselves mature professionals, if indeed we are professionals at all?

Finally, why do we continue to tolerate those who use the label of professionalism itself to attack any serious attempt to achieve such? These people become especially agitated by any hint of assessment, certification, licensing, or any other potential measure of their credentials, yet we tolerate their frequent attacks of those seriously attempting to establish legitimacy for that which these critics take for granted for themselves. Their arguments against an elite class of practitioners also appears weak. If being elite means being concerned, involved, improving ones knowledge and skills, and contributing to the profession, then hooray for the elites. The cry of "ego trip" is no more valid for the achiever than it is for the non-achiever. Is it not much more ego involved to attribute characteristics to oneself without the verification of peer group evaluation?

What is suggested is that the industry begin to show maturity in at least three ways: First, that all assist in obtaining credentials reflecting clear definitions of distinguishing professional characteristics. Second, participate in implementing and acquiring legitimacy for these credentials. Third, cease to tolerate the distracting and demeaning attacks upon efforts to develop and mature the profession by actively responding to those who would diminish or destroy the efforts to improve our industry and performance within it.

Surely, tradition, common sense, and responsible self-interest dictate the formation of such a recognized computing profession.

The decision rests with us in the industry. All are urged to become concerned, to get involved, and to make commitments such that many voices and talents will be engaged in finding solutions to the challenges of the future toward the profession.

In this spirit remember the visionary words of George Bernard Shaw as he wrote:

> Some men see things as they are,
> and ask why?
> I dream of things as they might be,
> and ask why not?

REFERENCES

1. "Called Non-Professionals, Programmers Win Overtime," *Computerworld*, Vol. X, No. 6, p. 1, February 9, 1976.
2. Casper, G. Gary, "Privacy and the Licensing of Data Processing Personnel: Social and Ethical Issues," *Conference Proceedings SCDP-I*, Society of Certified Data Processors, November 22, 1974. Also *Conference Proceedings ACPA-IV*, Association of Computer Programmers and Analysts, November 27, 1974.

3. Casper, G. Gary, "Professionalism: Regulation, Voluntarism, Licensing, and Legitimacy," *Conference Proceedings ACPA-V,* Association of Computer Programmers and Analysts, October 1, 1975.
4. Casper, G. Gary, "The Licensing Controversy and It's Meaning To Professionalism," *Conference Proceedings of the Association for Educational Data Systems* (AEDS), Phoenix, Arizona, April 1975.
5. Casper, G. Gary, "Licensing the Data Processing Professional—Pros and Cons," Unpublished paper presented to the Data Processing Management Association Info-Expo, Las Vegas, Nevada, October 1976.
6. Casper, G. Gary, "A Maturing Profession," Unpublished paper presented to the Data Processing Management Association (DPMA) Info-Expo, Washington, D.C., October 10, 1977.
7. *Computerworld,* February 10, 1971.
8. Data Processing Management Association, "Software is Intangible Personal Service," Position Statement, August 1977.
9. Fierheller, George, "The Newest Profession," Unpublished paper presented to the Quebec Chapter of the Canadian Information Processing Society (CIPS), April 1974.
10. *The American College Dictionary,* Random House Inc., 1956.
11. Toffler, Alvin, *Future Shock,* Random House, Inc., July 1970.

Designing and debugging careers for women in the computer industry

by THELMA ESTRIN
Brain Research Institute, University of California
Los Angeles, California

Women's pursuit of careers in the computer industry is consistent with our national commitment to equal opportunity and with our national need to utilize all available talent in support of scientific research and industrial efforts.

Computer science education and immediate employment on the first steps of the career ladder are readily available for women but after these initial steps women still face cultural, educational and institutional barriers which block them from advancement to the top. Women are finding, as men have, that superiority in technical knowledge and competence are not sufficient to guarantee advancement. In addition, women must carefully plan their careers and remain alert to those societal attitudes, the hidden "bugs", that prevent them from successfully implementing their long-term career goals.

The concept of career planning is receiving much attention by professionals in all fields, but the rapid rate of technological and methodological innovation in the computer industry gives special urgency to the need for computer professionals to use planning techniques to combat technical obsolescence and to aid upward mobility. Career planning deals with the identification of ones goals, skills, and priorities and the designing of a career with alternative options. It involves deciding if one wants to pursue a technical or a managerial career and poses questions such as: What type of a job do I want in the next years? What is the knowledge base? What are the skills needed and at what level of competence and experience? What types of inter-personal relationships are important?

Many factors that relate to professional development apply equally to men and women but career planning is especially important to help compensate for the distinct problems women encounter with upward mobility, and for women who want to combine career advancement with family life. There are very few women at the top in any business or profession and the lack of role models serves to discourage women who might consider working towards leadership status in the computer field. Qualified women are often reluctant to move ahead because upward mobility requires significant deviation from social norms. In other cases, women may be hindered by the attitudes of men, who are their peers and supervisors. Withholding promotion of women may not be overt, but due to unconscious discrimination based on sex-role stereotyping and societal conventions.

The panelists will discuss various approaches to career planning. The opening panelist, a career development consultant, will present a model of career stages based on accepted theories of adult development. The other panelists, computer professionals in the areas of software, hardware and technical marketing, will discuss factors and strategies which they have found useful in advancing through career stages in the computer industry. They will also discuss barriers to women's full and equal participation which should be considered in career planning.

This panel is not for women only and men are encouraged to attend. In addition to benefiting from those aspects of career planning which are of value to all professionals, men can gain insights to improve their working relations with female colleagues from an awareness of the special problems professional women encounter in pursuing non-traditional careers.

A panel session—Computers in early education

SESSION CHAIRMAN—ORLANDO S. MADRIGAL
California State University, Chico

Panel Members

Alan Kay—Xerox Palo Alto Research Center
David Moursund—University of Oregon
John Maniotes—Purdue University
William G. Lane—California State University
A. M. Banks—Beyer High School

PANEL OVERVIEW—Orlando S. Madrigal

This session will have five panelists/speakers who will discuss the use of computers at the pre-college level. The speakers will include topics related to:

- The cost-effectiveness of the use of computers for high school education, by William G. Lane—California State University, Chico.
- Computer Science education for preservice elementary school teachers, by David Moursand—University of Oregon.
- Programming for children on a personal computer. To include a brief movie that will demonstrate the speaker's experiences with children who are taught programming at an early age. By Alan C. Kay—Xerox Palo Alto Research Center.
- The teaching of micro-computers in high school, by Amberse Banks—Beyer High School, Modesto, California.
- The state of high school data processing programs, by John Maniotes, Purdue University.

SPEAKERS AND TOPICS:

(1) Professor David Moursund, University of Oregon
Title: Computer Science Education for Preservice Elementary School Teachers
Abstract: Calculators and microcomputers are now cheap enough so that elementary schools can afford to give students quite good access. Thus this equipment could be playing a major role in the curriculum. But progress to date has been slow. A major cause is lack of suitably trained teachers. Preservice elementary school teachers need to receive a substantial introduction to computer literacy, teaching using calculators and computers, and teaching about calculators and computers. The author has been involved in the development of courses of this sort, and is currently teaching such a course. This paper includes a detailed course outline and discussion. References to the text used, as well as supplementary material, are given.

(2) Professor John Maniotes, Purdue University, Columett Campus
Title: The Interface Between High School Data Processing Students and the Two-Year Community College Programs in Computer Science
Abstract: Data and problem areas will be presented to explore the transition of high school students, with data processing background, to a formal 2-year curriculum in Computer Science. More and more high schools teach computer-related courses. Students in these programs are able to phase into a college program in computer science including the exemption from the basic courses including introductory programming and concepts. A high school data processing program that is well articulated with college level programs will greatly benefit the students in the program. There will be less course duplication and students will be able to obtain advance placement in the Computer Science area.

(3) Professor William G. Lane, California State University, Chico
Title: Cost-Effectiveness of the Use of Computers For High School Education
Abstract: As more and more high schools start offering computer programming courses, the cost of high school education will greatly increase. And with the trend towards a "no increase" in school budgets, new avenues of educational financing will have to be pursued in order to allow school districts to pursue the addition of basic computer science to existing high school curricula. It will be shown that various avenues of cost-sharing among computer users in a given area can reduce the "per student cost" for computing use to a level that will be within the budgetary restrictions of most school districts. Another area that will affect cost is the need for high school teachers who can qualify as teachers in the Computer Science area. The

current move to establish high school credential programs in various states will be discussed.

(4) Dr. Alan Kay, Xerox Palo Alto Research Center
Title: Experiences With "Small Talk" and Pre-Schoolers

Abstract: The author has been involved with the development of personalized computing systems including both hardware and software. A major result of his work is the "Small Talk" system which utilizes a mini computer that can be used interactively to train preschoolers in the use of computers. The system utilizes a specially designed terminal that will allow the user to perform interactive color graphics without prior training in the use of computers. Hundreds of preschoolers have gone through the "Small Talk" system, and interesting observations have been noted including the users response to a unique computer environment and eventually relating these experiences to the classroom.

These systems, although not yet commercially available, can be considered as the forerunner to personalized computing where users can gain exposure to computing at an early age.

(5) Mr. Amberse Banks, Beyer High School, Modesto, California
Title: Augmenting a High School Curriculum With Microcomputer Education

Abstract: Microcomputer education can be paralleled to the traditional electronics training for television and stereo systems. However, microcomputers are more complex since computer programming must be included in the students work.

The author has devoted two years to the development of high school courses which train students in the intricacies of building and programming microcomputers. More and more students are getting involved in these courses. A number of these students have aspirations for college-level training in Computer Science and eventually enter the field as computer professionals.

PROGRAMMING FOR CHILDREN ON A PERSONAL COMPUTER—Alan Kay

The "Small Talk" system utilizes a mini computer with an attached terminal that can be used interactively to train pre-schoolers in the use of computers. The system utilizes a specially designed terminal that will allow users (preschoolers) to perform interactive color graphics without prior training in the use of computers. Hundreds of preschoolers have gone through the "Small Talk" system, and interesting observations have been noted including the users response to a unique computer environment and eventually relating these experiences to classroom learning.

These systems, although not yet commercially available, can be considered as the forerunner to personalized computing for children where users can gain exposure to computing at an early age.

COMPUTER SCIENCE EDUCATION FOR PRESERVICE ELEMENTARY SCHOOL TEACHERS—David Moursund

Calculators and microcomputers are now cheap enough so that elementary schools can afford to give students quite good access. Thus this equipment could be playing a major role in the curriculum. But progress to date has been slow. A major cause is lack of suitably trained teachers. Preservice elementary school teachers need to receive a substantial introduction to computer literacy, teaching using calculators and computers, and teaching about calculators and computers. This talk includes a detailed course outline and discussion of course requirements for such training.

THE STATE OF HIGH SCHOOL DATA PROCESSING PROGRAMS—John Maniotes

In recent years, computers have had a profound effect on all of the institutions of society, including early (and late) education. If we, as educators, are to prepare students for the future, we need to participate fully in the computer evolution/revolution and to teach our students to use, understand, and live with these logical machines that will exert such a significant influence on the rest of their lives.

The number of computer and data processing related courses and programs a high school has to offer depends upon the size of the school, the availability of experienced EDP or computer science teachers, the attitude of the school's administrators, and above all the amount of funds allocated to the EDP program. Although some schools continue to overlook the impact that computers have had in the business and scientific world, others have found economic ways to incorporate some kind of computer-related courses in their curriculum.

Several programs that have been categorized according to the equipment available to the students are described. These programs include the:

- Textbook concept programs
- Input/Output terminal programs
- Centrally located computer concept programs
- Home-based computer concept programs

For those schools which lack the aforementioned programs, all sorts of extracurricular activities can be found intermingled with the obligatory academic subjects. In particular, the role of the ACM computer clubs, science and math clubs, and the explorer posts are described.

Finally, some of the major problems facing high schools in implementing vocational data processing programs are

explored. In particular the problems associated with:

- Curriculum objectives and guidelines
- Recruitment of qualified instructors
- On-going teacher education in EDP
- Instructional materials
- Computer oriented aptitude tests
- Funding to support the EDP programs
- Physical facilities for computer hardware
- Computer software and application packages
- Recent impact of microcomputers, intelligent terminals, and programmable calculators
- Interfacing with 2-year BDP and 4-year CS programs at colleges and universities.

COST-EFFECTIVENESS OF THE USE OF COMPUTERS FOR HIGH SCHOOL EDUCATION—William G. Lane

A decade of program development in elementary and secondary education has been characterized by oversell, underdelivery and poor timing. CAI projects, which promised to raise the overall class growth "X" grades, while letting everyone proceed at his own pace, ran into a number of problems:

- Parent taxpayers, whose only probable previous experience with the computer was in trying to get a credit card purchase error corrected, raised much concern over "turning the education of our children over to a computer."
- Teachers were resistant to adopting new methods that too frequently were perceived as having a potential for lowering the number of faculty positions and that had the effect of reducing their status to a regulated learning manager rather than an independent learning deliverer.
- Curriculum personnel and school administrators were asked to operate in a new environment for which they had little or no training.
- Marketing analysts had underestimated the cost and length of the development effort and over-estimated early market, causing computer companies to become disenchanted and to withdraw support.
- Low cost hardware technology required for broad acceptance had not yet been developed.

Given these problems, it is not surprising that computers in the elementary secondary education areas have not even begun to reach their potential. But times have changed; hardware costs have lowered, TV games and microprocessors have become a major factor in the home market, the computer is no longer perceived as a job-eating monster and, as such, we are now probably ready to again access the feasibility of installing computers at all levels of our school system.

Several pilot studies are evaluated and analyzed for their potential:

- The use of computer graphics in the teaching of mechanical drawing at the junior high level.
- The use of modularized instruction and computer-based testing in lower elementary levels.
- The use of a computer in handicapped education.

Each of the above studies showed a need for differing systems and peripheral architectures, as well as responsibility for the classroom teachers. These led to the requirement for differing architectural concepts and approaches.

TEACHING MICRO-COMPUTERS IN HIGH SCHOOL—A. M. Banks

In view of the impact of computers upon our society, few people would question their inclusion in the secondary school curriculum. But why should it be done by using micro-computers? Can such curricula be of value to students? While it is true that such small systems do have more limitations than do larger systems, they also offer many new options to the high school program that only have access to micro-computer systems.

Most computer courses in high schools are arranged to suit the type of equipment that is available to the students. Terminals are the most common form of computer access at the high school level; however, in many cases, the students are assigned as low priority users. The restrictions that are thus placed on high school programs curtail, prevent, or even discourage students. Also, the increasingly bleak financial situation of most school districts makes the acquisition of systems large enough to support more sophisticated time-sharing a virtual impossibility. Also, the rising costs of telephone lines makes the use of district-wide systems more and more difficult.

Highly flexible, expandable micro-computers now cost the same as terminals did a year ago. The newer self-contained, assembled systems cost so little that shcools can afford to build up a laboratory with several machines. Multiple machines offer variety and protection against system-wide downtime.

Can the computing work that is possible on a micro-computer compare with that possible on larger systems? Yes and no. First there is the matter of the languages that are available. Currently, there are several excellent BASIC interpreters and some reasonable assemblers available for each of the microprocessors used in micro-computer systems. In addition, there are versions of PILOT, APL, Pl/1 and FORTH available for some chips. While the lack of COBOL, for example, is a handicap if we are training professional programmers, these available languages are more than adequate for high schools. Software advances are being announced every month for those who do not wish to write their own. This is an area where interested students can do advanced work on micro-computer systems but not in larger systems.

There is an additional aspect of computers that can be covered using micro-computers which cannot even be attempted with larger systems. It is possible to have students do hardware work, including maintenance and hardware configuration. It helps the student to view a computer as a tool to be used rather than as a shrine into whose presence only a privileged few are permitted. Instead, they find new ways and new places to put it to work. This kind of attitude is not generated when students use larger computers or even mini-computers.

Thus, we see that micro-computers can match almost every service offered by a larger system to high school classes, and can offer additional opportunities to students that are just not feasible with larger systems.

Computer education in higher education—status, alternatives and needs

by JOHN W. HAMBLEN
University of Missouri-Rolla
Rolla, Missouri

INTRODUCTION

The cause of many of the problems associated with computer usage is the "overutilization of undereducated people." The reason being, of course, that properly educated people have not been available. Bootstrapping by training existing personnel and pirating whatever other centers had trained was the only way that staff could be obtained during the late fifties and the early sixties. The late sixties saw the tremendous growth in the one and two-year programs aided by large infusions of federal monies via the Office of Education programs. The production of these programs crested around 1972 at a peak of nearly 28,000 during that year. The seventies might well become known as the decade of the recognition of the value of the college graduate to effective and efficient computer usage. Based upon my estimate of need and the present average rate of growth (about 10 percent) in the production of graduates of four-year programs it would take nearly twenty-four years to match the need and production.

The main point here is that it will be some time before we need to worry about a "glut" in the market for four-year graduates in the computer related areas. At the master's level, the rate of growth is about half that of the four-year production rate and it will take nearly twice as long to match production with need. At the doctorate level fifteen years will be required to match production with need based upon my estimates of need and current rates of growth in the doctorate production.

All of my estimates are based upon the expectation that the professors in colleges and universities will seek out the advice of representatives of industry as to industries' needs and keep the educational programs aligned with those needs. They cannot afford to do otherwise.

In the following I present my estimates[7,9] of computer manpower based upon the best, though limited, data available classified by job classifications and level of education which I believe will be required for replacement of current personnel and to fill new positions. Better data is available on manpower production than on utilization through the efforts of SREB Computer Science Project[1-3] and the American Federation of Information Processing Societies[4,5] with support from the National Science Foundation. Data from the Bureau of Adult, Vocational, and Technical Education of the U. S. Office of Education[6] are also utilized. Manpower production is then matched against estimated needs.

Although the need for computer manpower education is great, there is at least an equal need for most people to become "computer literates" as a minimum. This can be easily accomplished if all teachers and other college graduates can become a little more than just "Computer Literate" (e.g., at least one course involving some programming).

BACKGROUND

During the period 1966-72 while I was Project Director for Computer Sciences at the Southern Regional Education Board (SREB), I conducted three surveys on Computers in U. S. Higher Education including their utilization and related educational programs. These studies were supported by the National Science Foundation and resulted in three publications.[1-3] The first was published by the SREB and the latter two by the National Science Foundation.

The first study was a survey using a stratified random sample of 739 institutions, from a population of 2219, selected by systematic random sampling within strata by the U. S. Office of Education. Six hundred sixty-nine or 92 percent of the institutions in the sample responded.

The base year for data collection was 1964-65 and projections were requested from the institutions for 1968-69. The data collection instrument was designed by the Mathematical Sciences Section of NSF to meet their existing program needs for planning. Consequently, emphasis was placed upon the financial aspects of college and university computer center operations and only limited information was gathered with regard to manpower training and utilization.

The second survey was much more comprehensive and was sent to all institutions of higher education listed on the U. S. Office of Education's Higher Education General Information Survey (HEGIS) file. Nineteen hundred sixty-five or 79 percent of the 2477 institutions responded. The data collection forms were designed with the advice and counsel of thirty or more national professional and institutional associations and government agencies. The base year for reporting data was 1966-67.

Extensive modifications were made in the data collection forms for the third survey. The base year was 1969-70.

In 1972 I was commissioned by the SREB to do a paper on Computer Manpower—Supply and Demand—in SREB states. Using this as a starter, I prepared a similar report for all states.[7,9] Manpower production estimates were based upon the NSF supported surveys[1-3] and data from the USOE.

During 1974, I sent questionnaires to 2,520 institutions of higher education (seminaries, conservatories and special schools were not included) to update the data obtained on degree programs in the 1969-70 survey. A 61 percent rate of response was obtained and the production estimates for 1973-74 and 1974-75 were added to previous estimates.[7,9]

Another survey covering the 1976-77 year is nearing completion and comparable data will be supplied at the NCC in Anahein for all tables in this paper.

COMPUTER EDUCATION LEVELS AND ALTERNATIVES

It is generally recognized that there is a definite need for some degree of computer education for all college graduates. This can be accomplished at several different levels.

- Non-Credit Survey or Programming Short Course—This may consist of 8-16 hours of lectures and demonstrations and should be provided for faculty and staff on a continuing basis. Response by students is usually spotty. Very poor coverage can be expected from this approach.
- A Unit on Computers and/or Programming Introduced in a Required College Course—This is the best way to insure a minimum introduction to computers for all students. Faculty can be trained for this teaching through short courses, provided good training outlines and materials are made available to them.
- Integration of Computer Techniques as Tools in Most Disciplines—This part of the program requires an enlightened faculty. It will be several years before faculties exist which are able to do this on a college-wide basis. However, there are several fields in which this can be accomplished now. Among them are business, economics, statistics, mathematics, physics, chemistry, engineering and the social, behavioral and biological sciences. If handled properly, the use of the computer can serve to motivate the student and to permit the instructors to use more realistic examples in their instruction.

TABLE I.—Numbers of Colleges and Universities Offering Computer Appreciation Courses with Enrollment

	1969-70	1976-77 (Est.)
No. Institutions	152*	300
Enrollment	18112	50000

* Fifty-six other institutions reported such a course on the books but not yet offered during 1969-70.

TABLE II.—Courses and Enrollments in "Introductory Courses in Some Computer Language such as FORTRAN, PL/1, BASIC, ALGOL"

	1969-70	1976-77 (Est.)
Courses (Institutions)	971*	2,000
Enrollment	132,462	300,000

* Two hundred and six other courses were reported as being on the books but not offered yet during 1969-70.

COMPUTER APPRECIATION COURSES

Courses, such as Computer Appreciation, Computers in Society, Computers and Society, etc., have been popular for some time at many colleges and universities as is shown in Table I.

INTRODUCTORY COURSE IN SOME COMPUTER LANGUAGE SUCH AS FORTRAN, PL/1, BASIC, ALGOL.

These courses should consist of a minimum of two hours of lecture and two hours of laboratory each week. If handled carefully with regard to problem selection, this kind of course could be conducted as one multi-section course. However, better results can be obtained if some stratification is introduced when more than one section is needed. Such a course requires an instructor who has had some experience in programming in several computer languages. These courses will be required in some curricula and elected in others. The numbers of institutions (possible duplications) offering such courses and their enrollments are shown in Table II.

VOCATIONAL TRAINING PROGRAM

This is usually a one-year or two-year post high school curriculum. Several vocational training institutes initiated such curricula with federal and state assistance under the NDEA Title VIII program. Table III shows the estimated number of two-year programs offered and the number of degrees awarded.[7,9] Fifty-five (55) of these programs were in four-year and higher institutions offering the associate degree in data processing, computer programming, or computer technology.

TABLE III.—Estimated Number of Two-Year Degree Programs and Number of Degrees Awarded

	1966-67	1969-70	1973-74	1976-77
No. Programs	178	405	338	400
No. Degrees Awarded	1281	6051	4831	7000

TABLE IV.—Estimated Number of Bachelor's Degree Programs and Degrees Awarded

	1966-67	1969-70	1973-74	1976-77
No. Programs	90	198	302	400
No. Degrees	594	2208	5951	9000

TABLE VI.—Estimated Number of Ph.D. Programs and Degrees Awarded

	1966-67	1969-70	1973-74	1976-77
No. Programs*	58	98	90	100
No. Degrees	174	307	369	500

* Includes options in Math, Electrical Engineering, etc.

FOUR-YEAR DEGREE PROGRAM FOR COMPUTER PROFESSIONALS

Although there is now a recognized need for many majors in computer science, information systems, etc., not many colleges can afford to properly staff such a program for several years to come. For one thing, the competition is very keen for new Ph.D.'s in Computer Science and very few exist in Information Systems.

One hundred thirteen bachelor's degree programs in Computer Science were reported in operation with 8000 majors and 1537 graduates for 1970-71. In addition, another 69 bachelor's programs were reported going with nearly 7000 enrollees and 1000 graduates. The latter included 28 programs in Data Processing, Information Sciences, and Computer Science as options in various academic areas, with the most being in mathematics and electrical engineering. All 28 of these programs were offered in institutions granting a master's degree or higher. Table IV gives estimates[7,9] of the numbers of bachelor's degree programs and the number of degrees awarded.

MASTER'S DEGREE PROGRAMS

A perennial debate is whether the M.S. degree should be professional or research oriented. Where possible those who want to go on for a doctorate should do a thesis option otherwise little distinction should be made. The number of M.S. programs and degrees continue to grow as can be seen in Table V.

DOCTORATE PROGRAMS

Conte and Taulbee[8] are gathering information annually on the "Production and Employment of Ph.D.'s in Computer Science" as a Subcommittee on Manpower of the Computer Science Board. Prior to 1976 their data was collected only from Ph.D. granting departments of Computer Science, Information Science, etc. Sixty departments reporting for 1976 graduated 246 Ph.D.'s and estimated that 310 would complete the degree in 1977. Considering the number of departments reporting these numbers are well in line with my estimates given in Table VI.

TABLE V.—Estimated Number of Master's Programs and Degrees Awarded

	1966-67	1969-70	1973-74	1976-77
No. Programs	98	155	157*	200
No. Degrees	742	1442	2854	4000

* 192 for 1975-76.

MANPOWER NEEDS

In References 7 and 9, I present a model which I developed to produce estimates of computer manpower needs. The results are given in Tables VII and VIII.

The post-secondary vocational institutions' production figures are added to the associate degrees to obtain the final "post-secondary and two-year" estimates.

These estimates say that we are close to producing enough two-year/post-secondary graduates in the computer field, but only one-half of the needed doctorates, one-eighth the number of bachelor's and one-eleventh the number of master's levels needed. This reinforces my earlier statement that we are overutilizing undereducated people in the computer related fields. Others may wish to take exception to the assumptions which led to these results. If so, I hope that I have made the procedures transparent enough in Reference 7 that they can produce their own estimates. Even if my estimates are off by 50 percent, we still would have severe shortages indicated at the four-year and master's levels for some years.

The master's degree, quite common in the management analysts and systems programmers categories, and not too uncommon among applications programmers. Some top supervisory positions in operations attract M.S. graduates in the computer and information sciences.

Table VII gives my estimate of the distribution of computer staff by level of education that I believe to be desired for replacement if an adequate supply of manpower is available at these levels. During the past years, companies have had to take what staff they could get and train or retrain them. This is an expensive route and therefore, I believe that given a choice, industry would prefer to hire trained personnel. In terms of formal education, I believe that we are overutilizing undereducated people in the computer related occupations.

COMPARISON WITH BLS ESTIMATES AND PROJECTIONS FOR 1980

In the 1972-73 Occupational Outlook Handbook published by the Bureau of Labor Statistics, it was reported that there were over 100,000 systems analysts in 1970, over 200,000

TABLE VII.—Estimated Distribution of 1973 Computer Staff by Level of Education Required or Desired for Replacements

	PS VOC and 2-year	4-year	M.S.	Ph.D.	Total	Percent of Total
Teaching	200	1,500	2,000	2,200	5,900	.6
Management	3,000	37,000	60,000	5,000	105,000	11
Analysts	1,000	62,000	100,000	2,000	175,000	18
Systems Programmers	2,000	33,000	70,000	—	105,000	11
Applications Programmers	10,000	200,000	35,000	—	245,000	25
Operators	100,000	100,000	10,000	—	210,000	22
Other Operating Personnel	100,000	12,000	—	—	112,000	12
Total	216,200	445,500	287,000	9,200	957,900	
Percent of Total	23	46	30	1		

programmers and over 200,000 operators. Assuming a 10 percent annual growth factor (Gilchrist, Computerworld, McGovern of International Data Corporation, and U.S. Department of Labor in the U.S. Economy in 1980, Bulletin 1673, p. 47) we can expect these numbers to double by 1980. This extrapolation would give 200,000 systems analysts, 400,000 programmers and 400,000 operators for 1980. However, the 1970 estimate for systems analysts does not agree with the 1968 figures given in the U.S. Economy in 1980, p. 59, which gives 150,000 systems analysts for 1968 or when extrapolated to 1970, 180,000 systems analysts for 1970 and 240,000 for 1973, as compared to my estimate of 175,000. The projections for 1980 are presented on the same page and are 425,000 for systems analysts, 400,000 for programmers and 400,000 for operators. Some of the differences result from definitions. I would guess that some of what I call systems programmers are counted by BLS as programmers and others as systems analysts since no such category as systems programmers appears in BLS publications referred to above. It is of interest to note (p. 59, also) that systems analysts, programmers and computer operators were expected to be the fastest growing occupations in the 1970's.

Management personnel are most likely to be counted under "Managers" in the BLS classifications and on page 293 of the 1972-73 Handbook there is an indication that what I call other operating personnel (excluding card punch operators) are included under Office Machine Operators. In particular, operators of tabulating machines and related equipment, where such equipment is an integral part of a computer installation, would fall under my classification of "other operating personnel."

In BLS Bulletin 1673 referred to above it was estimated that there would be 120,000 computers in use by 1980 and that the number of process computers would reach about 17,000. I believe that the 120,000 figure is conservative, depending upon one's definition of a computer. In the same publication, it was estimated that for the period 1968-80 there would be annual openings for 23,000 programmers, 27,000 systems analysts and 20,400 operators, or a total of 70,400 in just these three classifications. Since these three categories account for about two-thirds of my total estimates, extrapolation of the 70,400 figure gives about 105,000 annual openings per year. This number compares very well under the circumstances with my estimated need of 117,000 per year.

TABLE VIII.—Future Computer Manpower Needs Per 1000 Estimated 1973 Personnel

	Education Level			
	PS Voc and 2-yr.	4-yr.	M.S.	Doctorate
Cause:				
Replacement:				
Deaths	5	5	2	2
Retirements	17	17	17	17
Growth	100	100	100	100
Total Rate	122	122	119	119
Total New Openings	26,376	55,571	34,153	1,095
Estimated 1975 Productions	19,286	6,996	3,236	493

COMPARISON WITH HEGIS DATA

Gilchrist and Weber[5] refer to classification differences which exist between the HEGIS data and the data which I collected at SREB. The main reason for differences in number of graduates reported is that HEGIS data, in general, reflect graduates of so-named departments only. For example, graduates of mathematics departments with computer science options would be classified as mathematics (or a subheading) and not computer science. The same would, in general, be true of graduates of other departments (EE, IE, etc.) with computer science options. Likewise, graduates of departments in business with options in systems analysis would not be reported under systems analysis. For this reason, the data reported in the inventory conducted by me at SREB and my updates for 1973-74 and 1974-75 should be considered to reflect more closely the actual numbers of graduates being produced in these computer related fields.

SUMMARY

Large numbers of personnel have been able to enter the computer manpower pool in the past with little training. The

proprietary schools had a "heyday." Only the best and largest have survived. Large amounts of federal and state funds were poured into the vocational and two-year programs. I question the value of those operated at the secondary level and have indicated[7,9,10] that there should be a weeding out of the weakest post-secondary vocational and two-year programs. I differ with Gilchrist and Weber[4,5] in that I do not think that secondary schools will be significant producers of computer manpower. On the other hand, computer education at the secondary level should not be neglected. Secondary schools should provide enough computer education so that the student can learn to live comfortably in a computer assisted environment and also enough so that students pursuing post-secondary education can judge whether or not they wish to pursue a program of study which will prepare them for entry into the computer manpower pool.

REFERENCES

1. Hamblen, J. W., *Computers in Higher Education: Expenditures, Sources of Funds and Utilization for Research and Instruction 1964-65 with Projections for 1968-69,* Southern Regional Education Board, Atlanta, Georgia 30313, 1967.
2. Hamblen, J. W., *Inventory of Computers in U.S. Higher Education 1966-67: Utilization and Related Degree Programs,* Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 1970, LC#74-609402 (out of print).
3. Hamblen, J. W., *Inventory of Computers in U.S. Higher Education 1969-70: Utilization and Related Degree Programs,* Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 1972, 401 pg. LC#74-609402 (out of print).
4. Gilchrist B. and R. E. Weber, "Employment of Trained Computer Personnel—A Quantitative Survey," *Proceedings of 1972 SJCC,* American Federation of Information Processing Societies Inc., Montvale, N.J.
5. Gilchrist, B. and R. E. Weber, "Sources of Trained Computer Personnel—A Quantitative Survey," *Proceedings of 1972 SJCC,* American Federation of Information Processing Societies Inc., Montvale, N.J.
6. ——, *Summary Data—Vocational Education Fiscal Year 1973,* Vocational Education Information No. I; U.S. Office of Education, May 1974.
7. Hamblen, J. W., *Computer Manpower—Supply and Demand—By States;* Information System Consultants, St. James, MO, 1973, 1975, 39 pages.
8. Conte, Sam D. and Orrin E. Taulbee, "Production and Employment of Ph.D.'s in Computer Science," Annual Survey for Computer Science Board, University of Pittsburgh.
9. Hamblen, J. W., "Computer Manpower in the U.S.—Supply and Demand," *SIGCSE Bulletin,* Association for Computing Machinery, New York, NY, December 1975, December 1977 (an update).
10. Hamblen, J. W., "Using Computers in Higher Education: Past Recommendations, Status and Needs," *Communications of the ACM,* Association for Computing Machinery, New York, N.Y., Vol. 14, No. 11, pp. 709-712, November 1971.

Computer science and computer engineering— A review and overview of curriculum development

by GERALD L. ENGEL
Old Dominion University
Norfolk, Virginia

and

OSCAR N. GARCIA*
National Science Foundation
Washington, DC

INTRODUCTION

The introduction of the digital computer into the academic environment created a situation that would lead to impact on how teaching and research are conducted at virtually all levels, and in virtually all disciplines. The further combined evolution of semiconductor technology with software practice and methodology has resulted in a new field which has developed at an astounding rate. The impact of this rapid change in the academic environment has forced universities to modify their classic attitude of slow change on curricula and research matters to yield to external demands for education and research in computer related topics.

Though the implication of the computer has been present in all areas of the academic institution, this paper will be directed to education about the computer; that is, computer science and computer engineering. Within these areas the changes that have occurred can in some way be observed by the changes that have occurred in pedagogical hardware. Initially, most programs in these areas relied on a "large" central computer resource of the institution, and perhaps, a modest logic design laboratory. With their introduction, minicomputers were acquired in increasing numbers and it became common for a major department to have its own system. More recently, low priced microprocessor systems are placing significant computer power in the hands of almost any department.

An equally significant development went on in the classroom as in the laboratory. Initially, programming classes were presented often by the computer center staff. These quickly moved into more academic settings within mathematics and electrical engineering departments. From this point options and minors developed. These, in turn, developed into degree offering departments, which included in their offerings the tool courses but also moved up to more abstract courses, such as switching and automata theory, computability and formal languages.

In the sections that follow, the development of the academic concepts of this area of education will be considered, with primary emphasis on the more recent developments.

THE DEVELOPMENT OF CURRICULUM IN COMPUTER SCIENCE

The single most significant document in computer science education was the report of the Curriculum Committee on Computer Science (C^3S) of the Association for Computing Machinery, most commonly known as "Curriculum '68."[1] The report gives detailed specifications for twenty-two courses fundamental to the study of computer science, as well as putting these courses into a prerequisite structure that also involves mathematics.

While the questions of definition and justification of computer science are not addressed in the report, the subject matter is classified into three areas of computer science and two related areas. A core program is defined, and based on this core area the undergraduate program is detailed involving computer science courses, mathematics courses, and technical electives.

Graduate degree programs are mentioned but not detailed to the extent of the undergraduate program. The master's program is outlined in terms of areas while doctoral programs are only mentioned in general terms, with ideas put forth for further work. Additional topics which received some attention in the report included service courses and minors, continuing education programs, program implementation problems, and facilities.

"Curriculum '68" may be seen as an expansion and refinement of the 1965 report of C^3S[2]. This earlier report spent a great deal more time on the definition of computer science as a discipline, and its relationship to other disciplines, especially mathematics.

The 1965 report of C^3S was, in turn, an outgrowth of a panel presentation at the 1963 ACM national meeting that was reported in 1964 in *The Communications of the ACM*

* On leave from the University of South Florida, Tampa, Fla.

by Perlis,[3] Arden,[4] Forsythe,[5] Korfhage,[6] Gorn,[7] Muller,[8] Keenan,[9] and Atchison and Hamblen.[10] In these reports one finds a definition and specification of computer science; courses and program description emphasizing the first course, numerical analysis, logic and logic design; and descriptions on programs then existing in the United States. Within the courses and program descriptions emphasis was placed on the role of programming projects and exercises as an integral portion of the programs, and the close ties of computer science education to mathematics, at that time, was noted.

COMPUTER ENGINEERING

While the more theoretical aspects of computing machinery was being addressed by C³S, the improvement in economics in the computer industry opened new areas in logic design. Not only was computer design the main objective of studying combinatorial and sequential circuits, but also new areas in process control, data communications, systems analysis and simulation required this knowledge. A trend was initiated in colleges of engineering and particularly among electrical engineering departments to create the discipline of Computer Engineering.

In September 1967 the COSINE Committee of the Commission on Engineering Education published recommendations for an undergraduate course program.[11] Though originally designed as an enhancement of an electrical engineering curriculum this report was, in effect, the initial development of the area of computer engineering.

The report presents a series of subject areas in a computer science program in electrical engineering consisting of basic subject areas including programming principles, computation structures, discrete mathematics, and machines, languages and algorithms; and recommended elective areas including digital devices and circuits, switching theory and logical design, programming systems, numerical methods, optimization techniques, circuit and system theory, information theory and coding, functional analysis, combinatorics and applications, probability and statistics and symbol manipulation and heuristic programming.

The COSINE Committee did not regard its mission as the same as C³S and did not equate electrical engineering and computer science. Nonetheless, it is of note that much of the material recommended parallels that of "Curriculum '68." The main differences appear in the advanced areas, and in a lack of a course or area corresponding directly to compiler construction. The core of the two programs are virtually the same, however, the work reported on by the COSINE Committee does not place the area of data structures in the fundamental position given by C³S.

The COSINE Committee issued eight other reports dealing with the first course in electrical engineering,[12] computer organization,[13] digital subsystems,[14] industries' reaction,[15] a computer engineering option,[16] digital system laboratory courses,[17] operating systems,[18] and the minicomputer in laboratory programs.[19] Perhaps the lack of a stable operating organizational base such as a professional society prevented the COSINE reports from having the broad impact of the more integrated report of C³S. An effort following in the steps of the COSINE Committee has been the DISE Committee, funded by the National Science Foundation, and with some of the same problems. It should be noted, however, that the efforts of the DISE project are more directed at dissemination of materials than at the definition of programs.

CURRICULUM DEVELOPMENT SINCE "CURRICULUM '68" AND THE COSINE REPORT

Subsequent to the publication of "Curriculum '68" and the COSINE Report, considerable work has gone on relating to computer science education. A significant development in this period was the formation of the ACM Special Interest Group on Computer Science Education (SIGCSE), which provides a continuing organization for the presentation and exchange of ideas in the field.

C³S has continued its activities following the publication of "Curriculum '68." Under the sponsorship of this Committee, a series on doctoral programs was published, a summer institute program for smaller colleges was conducted, course recommendations for smaller schools were prepared,[20] and guidelines for masters programs were prepared.[21]

Additional curriculum work has been conducted within ACM by the Curriculum Committee on Computer Education for Management (C³EM). This Committee has prepared guidelines for both graduate programs[22] and undergraduate programs[23] in information systems which integrates materials associated with computer science with materials associated with business.

As was indicated in the previous section, the COSINE Committee was active in this period, and in addition to the publication cited, the Committee also conducted a series of workshops to assist in the interpretation of their work.

The Committee on the Undergraduate Program in Mathematics (CUPM), also published recommendations in the area of computational mathematics[24] during the period. This work represented an extension of earlier work describing a mathematics program with work in computing.[25]

In addition to these efforts, the literature of computer science education contains numerous references to specific courses and topics providing an expansion of the course material given in the curriculum studies, and presenting something of a dynamic updating of the recommendations. The more recent work of this type emphasizes the area of software engineering and has been addressed in a recent volume.[26] A comprehensive survey of the post "Curriculum '68" literature in computer science education has been published,[27] and thus will not be further addressed at this time.

NEW CURRICULUM RECOMMENDATIONS IN COMPUTER SCIENCE AND ENGINEERING

The Education Committee of the IEEE Computer Society has been meeting regularly for over two years to design a

curriculum which may serve as a model for both the computer science and computer engineering approaches. This work led to the publication of a report of the Model Curricula Subcommittee of the Education Committee in February 1977,[28] although a preliminary report on the work had appeared earlier.[29]

Two aspects of this report should be taken into account for its correct interpretation:

1. The total curriculum as presented probably exceeds the resources or time allocations of most undergraduate programs and, as a result, needs to be adapted to regional needs, local resources, and departmental orientation.
2. A central "core" of subject matter is outlined that is considered the kernel of an undergraduate program in computer science and engineering.

There is no pretention that the report is either exhaustive, complete or definitive. It is claimed that it does represent the views of representative segments of the educational community acting in a committee mode with suitable compromises.

The general distribution of the recommended course modules is divided into five areas as follows:

(a) Digital Logic—5 courses
(b) Computer Organization and Architecture—5 courses
(c) Software Engineering—8 courses
(d) Theory of Computing—4 courses
(e) Laboratories—7 courses

The titles of these courses are given in Figure 1. For catalog descriptions and more detailed information, the interested reader is referred to the full report referenced above.

It should be noted that from the number of courses, the software engineering area is the predominant area in the curriculum; thus, giving the curriculum a pragmatic and applied flavor. In the theory of computing area, a modest amount of course work is recommended which presents a number of very abstract principles in an introductory fashion.

Of particular note is the laboratory sequence. Most laboratories in present computer science and engineering programs, whether they are a part of a given course or a separate course, deal with a specific subset of hardware or

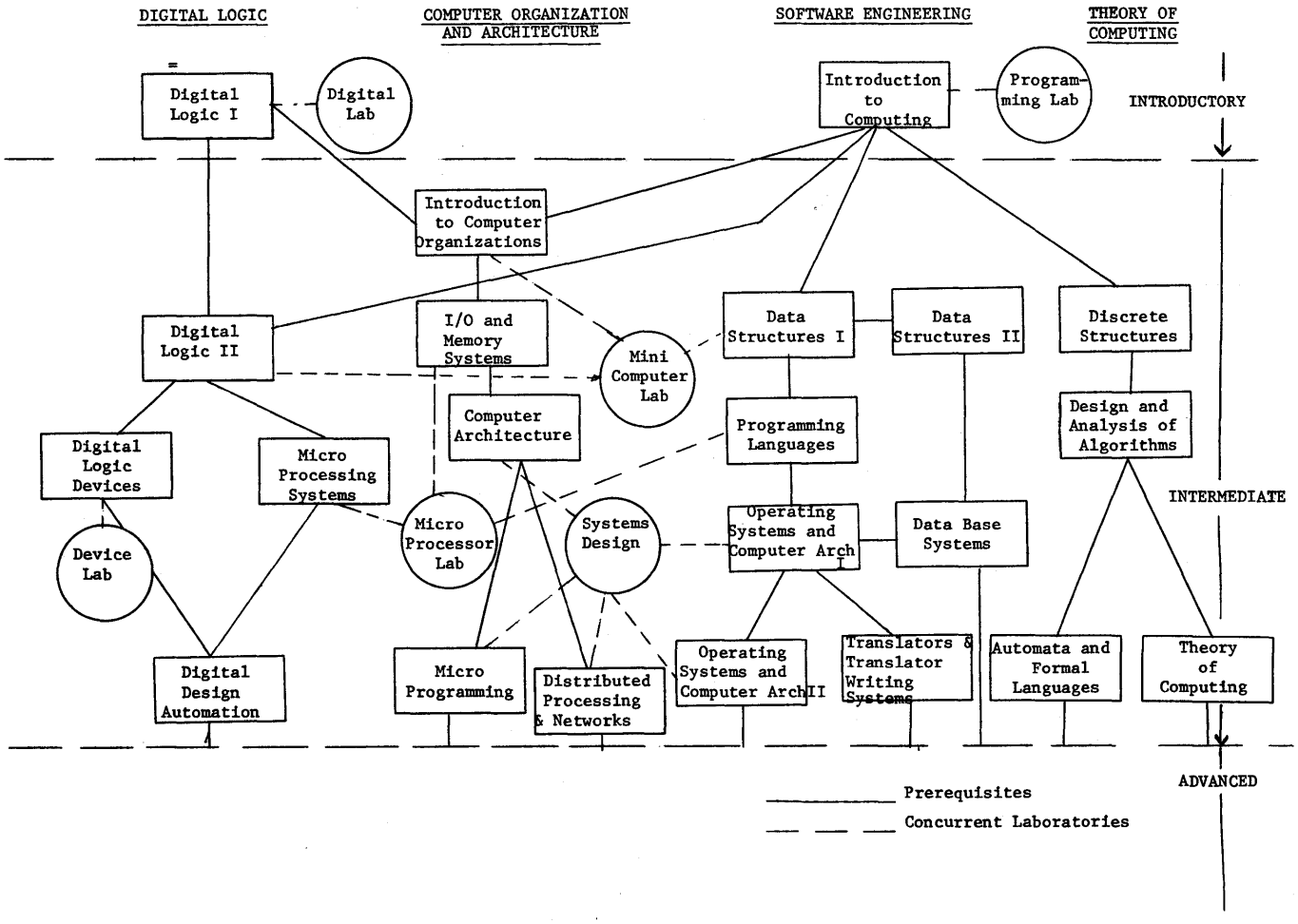


Figure 1—The courses of the IEEE/Computer Society curriculum recommendations

software problems. Therefore, a laboratory sequence should allow a student to become familiar with current technology to apply theory and to solve problems. Thus, the main objective is to expose the student to methods applicable to the "real world," to enhance creativity, and to develop problem solving skills. In the process, the student needs to learn a variety of techniques for designing, implementing, debugging, and maintaining a project.

The approach suggested here has a two-dimension organization.

1. *Vertically,* the labs are organized in a series of graded courses in which the student advances gradually from well-defined experiments to more sophisticated projects.
2. *Horizontally,* the labs are used as media to apply hardware and software principles in an integrated form.

The value of such experience becomes clear when the student realizes that the solution of a hardware design problem requires considering its software implications and vice versa.

The prerequisite structure for the program is given in Figure 2 which indicates the positioning of both the courses and the laboratories.

THE NEW RECOMMENDATIONS OF THE CURRICULUM COMMITTEE ON COMPUTER SCIENCE (C^{3}S) OF THE ACM

During the same period that the Computer Society was developing recommendations, C^{3}S was preparing an update and modification to "Curriculum '68." This appeared in draft form in June, 1977.[30] Though it will not be covered in this paper, it should also be noted that a subcommittee of C^{3}S prepared draft recommendations for a program at the two year level at the same time.[31] These reports were published in this way as working papers to obtain maximum input regarding their content prior to publication of the final committee reports which is anticipated in 1978.

The C^{3}S report is broken down into the general areas of the core curriculum, Computer Science electives, the undergraduate program, service courses and other considerations including facilities, staff and articulation.

The core material is considered to be at both the elementary and intermediate level. The material at the elementary level is first presented as a list of topics in the areas of programming, software organization, hardware organization and data structures and file processing. It is anticipated that many implementations are possible from this listing of topics, and it is emphasized that programming projects should be stressed throughout the course sequence. As a guide to implementation, but certainly not the only implementation, a five course sequence is proposed at this level: CS1-Computer Programming I, CS2-Computer Programming II, CS3-Assembly Language Programming, CS4-Introduction to Computer Organization, and CS5-Introduction to File Processing. Three intermediate level courses are then proposed to complete the core or that material required to provide minimum requirements common to all computer science undergraduate programs. They are CS6-Operating Systems and Computer Architecture I, CS7-Data Structures and Algorithm Analysis, and CS8-Organization of Programming Languages.

Following the core material illustrated in Figure 3, two intermediate level electives are proposed, as are nine advanced courses, and provisions for special topics courses. The course prerequisite structure in the report is given in Figure 3, and the interested reader is referred to the full report for more detailed information.

It should be particularly noted that the intermediate level core courses are recommended to contain a strong emphasis of fundamental concepts exemplified by various types of programming languages, architecture and operating systems, and data structures. Neither theoretical treatments nor case study approaches in and of themselves are adequate or appropriate at this level. Advanced level courses may be used for such treatments. Special topics courses are suggested and it should also be noted that which of the special topics are offered will be dependent on the resources of the department, however, it certainly is the case that in time, some of the material listed under this category, should be integrated into the courses more fully specified, or replace entire courses in the curriculum. Monitoring this phase of the program will be a continuing activity of C^{3}S.

Tied into the prerequisite structure of the program are six mathematics courses:

MA 1 Introductory Calculus
MA 2 Mathematical Analysis I
MA 3 Linear Algebra
MA 4 Discrete Structures
MA 5 Mathematical Analysis II
MA 6 Probability and Statistics

Descriptions of the courses MA1, MA2, MA3, MA5 and MA6 may be found in the 1965 CUPM recommendations.[32] MA4 represents a more advanced course in discrete structures than that given in "Curriculum '68," and builds on concepts developed in the study of calculus and linear algebra, emphasizing applications of discrete structures to computer science. The complete prerequisite structure is given in Figure 4 which includes all the courses mentioned, except CS 10, Computer and Society, and the Special Topics courses whose prerequisites will vary depending on circumstances.

The major will normally consist of the eight courses of the core material plus four additional courses selected from the recommended computer science intermediate and advanced electives with no more than two from any one specific subfield of the discipline. It is further recommended that the student take mathematics courses MA1, MA2, MA3, and MA4, and depending on computer science electives selected, MA5 and MA6 may be required.

As has been indicated, further details are available to the reader in the report, and in fact, this paper has only addressed those portions of the report dealing with the under-

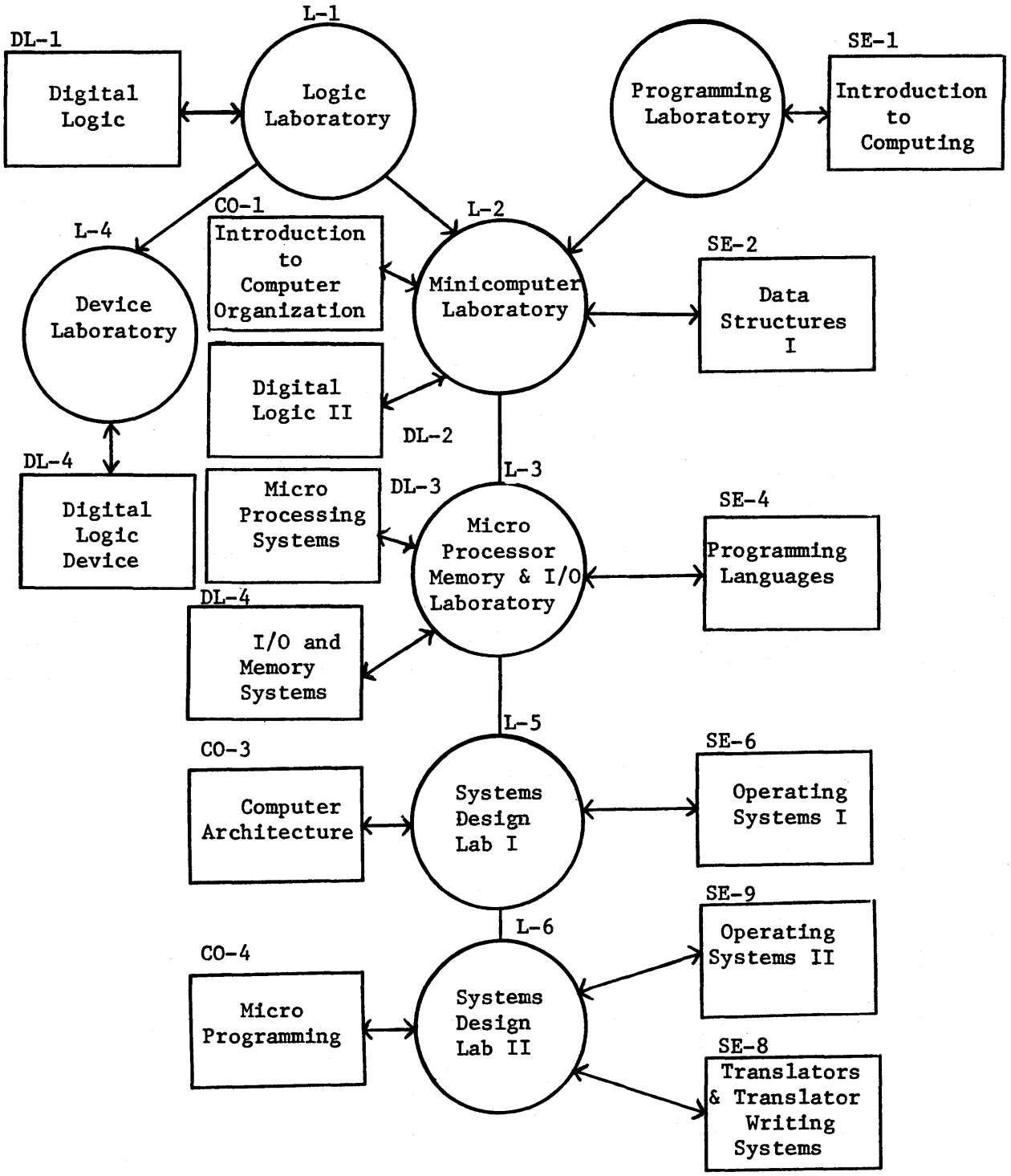


Figure 2—The prerequisite structure of the IEEE/Computer Society laboratory sequence

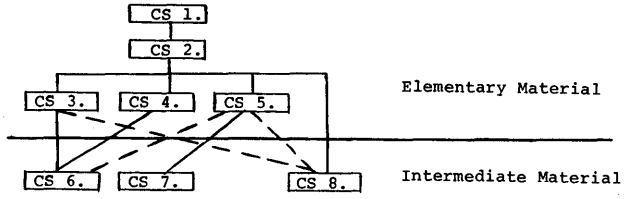


Figure 3—The core courses of the C³S recommended curriculum

graduate major. It should be noted, at this point, that the minimum requirement for the major, as specified, requires 48 semester hours which is certainly less than half the typical undergraduate program.

CONCLUSIONS, OUTLOOK AND PROJECTIONS

As has been shown in this paper, computer science and computer engineering have grown considerably in the past ten years. It would appear significant that it took ten years from the publication of the COSINE recommendations in 1967 to the publication of the model curricula of the Computer Society, while it also took ten years from the publication of "Curriculum '68" to the final version of the report updating that document.

What is perhaps most significant, however, is the convergence that can be seen occurring between the groups. Though working almost in total independence of each other, the two committees prepared guidelines remarkably similar in philosophy and content.

This similarity may be best seen in considering the core material of the two programs; CS1 Computer Programming I is similar to SE-1 Introduction to Computing, CS2 Computer Programming II is similar to SE-2 Data Structures 1, CS3 Assembly Language Programming and CS4 Introduction to Computer Organization cover much the same material as CO-1 Introduction to Computer Organizations, CS5 Introduction to File Processing is similar to SE-3 Data Structures II, CS6 Operating Systems and Computer Architecture I is similar to SE-6 Operating Systems and Computer Architecture 1, CS7 Data Structures and Algorithm Analysis covers much the same material as TC-2 Design and Analysis of Algorithms and SE-5 Data Base Systems, and CS8 Organization of Programming Languages is similar to SE-4 Programming Languages.

As one would expect, there are differences in the recommendations. The recommended core of the Computer Society program more strongly emphasizes the hardware and engineering areas, while those of C³S more strongly emphasize software and theoretical areas. But while these differences exist, the similarities strongly point to the fact that the discipline of computer science and engineering is approaching a well defined state.

It is fortunate that the two committees produced their reports at essentially the same time. This gives the educator the opportunity to study both documents, and select those parts that best fit his institutional needs in constructing a

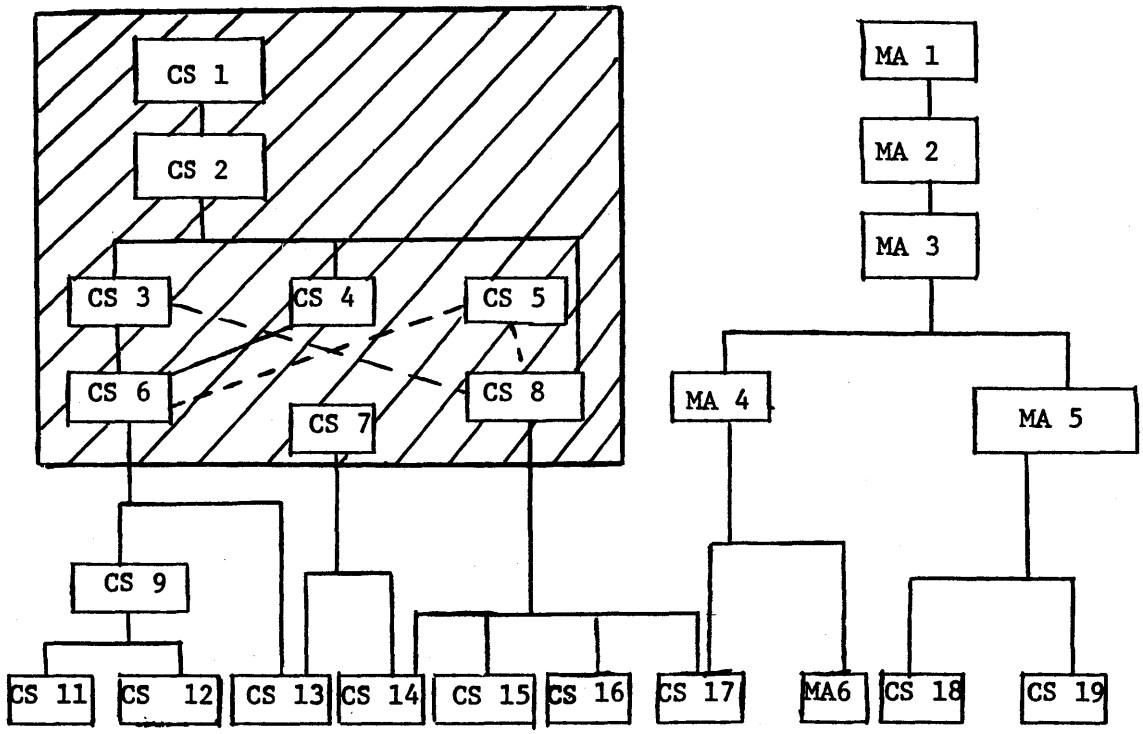


Figure 4—Prerequisite structure of the C³S curriculum

program, for it must be emphasized that these are only guidelines.

Coursework may be outlined on paper, however, more critical questions of feasibility of implementation in a variety of settings exists. While it is not anticipated that there will be complete agreement on courses and curriculum design on the part of all interested parties, by continued interchange of ideas, at least a consensus of the critical issues to be covered may take place. To foster this interchange, both committees welcome input from all interested parties.

REFERENCES

1. Curriculum Committee on Computer Science (C^3S), "Curriculum '68, ecommendations for Academic Programs in Computer Science," *Communications of the ACM* 11, 3, March 1968, pp. 151-197.
2. Curriculum Committee on Computer Science (C^3S), "An Undergraduate Program in Computer Science, Preliminary Recommendations," *Communications of the ACM* 8, 9, September 1965, pp. 543-552.
3. Perlis, A. J., "Programming Digital Computers," *Communications of the ACM* 7, 4, April 1964, pp. 210-214.
4. Arden, Bruce W., "On Introducing Digital Computing," *Communications of the ACM* 7, 4, April 1964, pp. 212-214.
5. Forsythe, George E., "An Undergraduate Curriculum in Numerical Analysis," *Communications of the ACM* 7, 4, April 1964, pp. 214-215.
6. Korfhage, Robert R., "Logic for the Computer Sciences," *Communications of the ACM* 7, 4, April 1964, pp. 216-218.
7. Gorn, Saul, "Mechanical Languages: A Course Specification," *Communications of the ACM* 7.4, April 1964, pp. 219-222.
8. Muller, David E., "The Place of Logical Design and Switching Theory in The Computer Curriculum," *Communications of the ACM* 7, 4, April 1964, pp. 222-225.
9. Keeman, Thomas A., "Computers and Education," *Communications of the ACM* 7, 4, April 1964, pp. 205-209.
10. Atchison, William F., and John W. Hamblen, "Status of Computer Science Curricula in Colleges and Universities," *Communications of the ACM* 7, 4, April 1964, pp. 225-227.
11. COSINE Committee, *Computer Science in Electrical Engineering,* Commission on Engineering Education, Washington, D.C., September 1967.
12. COSINE Committee, *Some Specifications for a Computer Oriented First Course in Electrical Engineering,* Commission on Engineering Education, Washington, D.C., September, 1968.
13. COSINE Committee, *An Undergraduate Electrical Engineering Course on Computer Organization,* Commission on Engineering Education, Washington, D.C., October 1968.
14. COSINE Committee, *Some Specifications for an Undergraduate Course in Digital Subsystems,* National Academy of Engineering, Washington, D.C., November 1968.
15. COSINE Committee, *Impact of Computers on Electrical Engineering Education a View from Industry,* National Academy of Engineering, Washington, D.C., September 1969.
16. COSINE Committee, *An Undergraduate Computer Engineering Option for Electrical Engineering,* National Academy of Engineering, Washington, D.C., January 1970.
17. COSINE Committee, *Digital Systems Laboratory Courses and Laboratory Developments,* National Academy of Engineering, Washington, D.C., March 1971.
18. COSINE Committee, *An Undergraduate Course on Operating Systems Principles,* National Academy of Engineering, Washington, D.C., June 1971.
19. COSINE Committee, *Minicomputers in the Digital Laboratory Program,* National Academy of Engineering, Washington, D.C., April 1972.
20. Austing, Richard H., and Gerald L. Engel, "A Computer Science Course Program for Small Colleges," *Communications of the ACM* 16, 3, March 1973, pp. 139-147.
21. Melkanoff, M. A., "An M.S. Program in Computer Science," *SIGCSE Bulletin,* 5, 1, February 1973, pp. 77-82.
22. Ashenhurst, R., (ed.), "Curricula Recommendations for Graduate Professional Programs in Information Systems," *Communications of the ACM* 15, 5, May 1972, pp. 363-398.
23. Couger, J. Daniel, "Curriculum Recommendations for Undergraduate Programs in Information Systems," *Communications of the ACM* 16, 12, December 1973, pp. 227-249.
24. Committee on the Undergraduate Program in Mathematics (CUPM), *Recommendations for an Undergraduate Program in Computational Mathematics,* CUPM, Berkeley, Calif. 1971.
25. Committee on the Undergraduate Program in Mathematics (CUPM), *Recommendations on the Undergraduate Mathematics Program for Work in Computing,* CUPM, Berkeley, Calif. 1965.
26. Wasserman, A. I., and P. Freeman (eds.), *Software Engineering Education: Needs and Objectives,* Springer-Verlag, New York 1976.
27. Austing, Richard H., Bruce H. Barnes, and Gerald L. Engel, "A Survey of the Literature in Computer Science Education Since Curriculum '68," *Communications of the ACM* 20, 1, January 1977, pp. 13-21.
28. Cain, J. T. (ed), *A Curriculum in Computer Science and Engineering: Committee Report,* IEEE Computer Society, Long Beach, Calif., 1977.
29. Mulder, M. A., "Model Curricula for Four-Year Computer Science and Engineering Programs: Bridging the Tar Pit," *Computer* 8, 12, December 1975, pp. 28-33.
30. Austing, Richard H., Bruce H. Barnes, Della T. Bonnette, Gerald L. Engel, and Gordon Stokes, "Curriculum Recommendations for the Undergraduate Program in Computer Science: A Working Report of the ACM Committee on Curriculum in Computer Science," *SIGCSE Bulletin* 9, 2, June 1977, pp. 1-16.
31. Little, Joyce Currie, Richard H. Austing, Harice Seeds, John Maniotes and Gerald L. Engel, "Curriculum Recommendations and Guidelines for the Community and Junior College Career Program in Computer Programming: A Working Paper of the Association for Computing Machinery Committee on Curriculum in Computer Science by the Subcommittee on Community and Junior College Curriculum," *SIGCSE Bulletin* 9, 2, June 1977, pp. 17-36.
32. Committee on the Undergraduate Program in Mathematics (CUPM), *A General Curriculum in Mathematics for Colleges,* CUPM, Berkeley, Calif. 1965.

The status of computer education in the community and junior colleges—Needs and alternatives

by JOYCE CURRIE LITTLE
Community College of Baltimore
Baltimore, Maryland

INTRODUCTION

The community and junior college is usually a multi-purpose, comprehensive, open-door institution, offering traditional transfer programs, career programs, continuing education courses, and community service courses. Each college is usually funded by both state and local taxes, and serves the citizens of a local geographic political division or sub-division. They offer access to higher education by means of generally lower tuition and by being within commuting distance of the citizens they serve. They are not often residential campuses, generally having a clientele who work part-time and maintain their role in their regular family situation.[1,2]

Although the first public community and junior college was established in 1901, it was not until the 1940's that the President's Commission on Higher Education called for making education more generally available through the establishment of two year colleges. Later commissions, in 1960 and 1970, recommended that a college be within commuting distance of almost every citizen of the United States. By serving as the "open-door" of opportunity to those who had no chance for higher education before, the two-year institutions have been credited as the "social leveler" in our society, and have brought this country closer to the goal of universal higher education.[2]

COMPUTER EDUCATION

The earliest programs in the computer field in the two-year colleges were those offered in California in the late 1950's. These early programs were generally built around unit record equipment and computers acquired through substantial discounts from IBM, in conjunction with support from the federal government. Because *business data processing* is the name generally used for record-keeping activities of a business or industry, most of the early programs carried this name and were offered through the auspices of the Business or Business Administration department. Many of these programs still exist, and often are named *Data Processing Technology, Information Systems, Computer Programming,* or *Computer Science.*

A typical career program in a two-year college offering an Associate in Arts or Sciences includes anywhere from 62 to 70 semester credits of coursework. This generally includes 24 credits of general education work, including English, Mathematics, and Social Studies, 12 to 15 credits in subjects in Business, and 18 to 24 credits of computer courses. Most of these programs generally include: three credits of an introductory data processing, three credits of an introductory computer programming language, as many as eight credits of COBOL programming, perhaps three credits of RPG programming, and six credits of systems analysis and design. Students sometimes have an opportunity to take one course in operations, and in some instances are included in an internship or work-study program. The mathematics requirement is generally either Business Mathematics or General College Mathematics, sometimes including a course in Elementary Statistics.

Additions to the offerings of the two-year colleges included, in many instances, a "scientific" option. These programs emphasize the use of the computer in the programming of statistical and data-gathering applications, in forecasting, and in mathematical and engineering work. Many of these programs are offered under the auspices of the mathematics departments; some of them are called *computer science.* More recently, two-year colleges near four-year programs have initiated the first two years of a transfer program, also generally entitled *computer science.* The scientific option generally differs from the business data processing or commercial option in these ways: classical mathematics up through the calculus is substituted for the courses in business, statistics is required, emphasis in computer language is on FORTRAN, and assembler programming is stressed in more depth. The transfer program, however, generally offers a course in data structures, one in computer organization or architecture, and a course in numerical methods.

A few schools offer an option within the Engineering or Electronics Departments. Students learn fundamentals in digital and analog computer logic and circuitry, and often

are employed by the manufacturers of computers to be further trained as customer service engineers.

Certificate programs are offered in some two-year schools. These are generally one year in length, but sometimes are one semester. Most computer operations certificate programs are one year, while certificate programs in data entry are generally shorter. In many states, these types of programs are offered by the secondary vocational-technical schools rather than in the community and junior colleges.

Many of the community and junior colleges offer courses about computers as a service to other departments and programs. Students majoring in Business or Accounting generally take one introductory course, as do students in medical records and paraprofessional health programs. Some schools offer "professional development" courses for industry, although the popularity of these courses has decreased at the community and junior college level as states have begun to limit the number of credits that may be transferred from these institutions to upper division colleges. Community service courses for no credit or for continuing education units (CEU's) remain popular with local community groups, especially within off-campus centers in local industry.

CURRICULUM GUIDELINES

Several curriculum guidelines for two-year programs have been prepared by the U.S. Department of Health, Education and Welfare Office of Education. A business data processing curriculum was prepared and released in 1963,[3] covering a broad range of careers in the computer field. In 1964, a report for electronic data processing in engineering, science, and business[4] gave suggested techniques for use in courses of study. A report on scientific data processing technology was released in 1970,[5] and in 1973, the earlier 1963 report was revised.[6] The 1963 publication was perhaps the most important influence on the establishment of programs during the 1960's.

In 1969, the American Association for Community and Junior Colleges (AACJC), sponsored the publication of two reports, with the assistance of a grant from Hewlitt-Packard Corporation. One concerned curriculum;[7] the other concerned college-wide computer usage.[8] There are currently no plans to update these reports by AACJC.[9]

A report giving recommendations and guidelines for a career program in computer programming for community and junior colleges was recently released by the Subcommittee for Community and Junior College Curriculum of the Association for Computing Machinery (ACM).[10] This report is based on the results of extensive discussion held between June 1975 and February 1977 with community college educators, industry representatives, and professional society representatives. The report appeared as a working paper in the SIGCSE Bulletin, along with a working paper for an undergraduate program in computer science, prepared by the Committee on Curriculum in Computer Science of the ACM, to receive reactions and comments before final publication.

This report represents the first set of curriculum recommendations to be produced for the community and junior college level by a professional society. In earlier ACM curriculum reports, however, providing educational programs for applications programmers is credited to the community and junior colleges.[11,12] The 1977 report is intended for the preparation of entry-level or trainee computer programmers who will work in an applications setting to support the general, administrative, and organizational functions of industry, commerce, business, and government service. Even though it is designed to prepare students for jobs, it emphasizes the need for a sufficient foundation for continued learning and advancement in the field. It is hoped that this report will encourage the re-evaluation of existing programs as well as serving as a guideline for the creation of new programs.

The ACM Community and Junior College Curriculum Report gives 14 objectives of the program, covering technical and non-technical skills. Instead of providing one sample or recommended curriculum structure, if offers guidelines for the local institution to adapt to the needs of their local community. It provides a detailed topical outline of the content of the program, and recommends that each college teach this content by giving depth instruction in a major procedural language (such as COBOL), some additional instruction in a second, or minor, language (such as RPG), and a foundation of concepts based on whatever assembler language it has available to use. It is recommended that the program include an "applied area" of study to be the major applications area. This feature of the curriculum recommendations allows the purpose of the program to be toward the commercial or business field, toward the scientific area, or toward any applied field of interest to the institution or to the student.

The report recommends laboratory experiences of a variety of types, with one type in one course, and perhaps another type in other courses. It is recommended that immediate turnaround of computer runs be provided during the scheduled laboratory sessions with the instructor present, with several additional turnarounds provided for open laboratory use. The report suggests an increased use of timesharing access, with experience in interactive programming provided. It recommends increased emphasis on concepts and usage of an operating system during coursework, increased emphasis on data and file structures, and increased use of pre-programmed packages and utilities.

The report offers a real challenge to those two-year institutions interested in maintaining an up-to-date, viable program. It will perhaps serve as an incentive to review and re-examine equipment, facilities, and programs.

EMPLOYMENT NEEDS

Students who complete the two-year Associate in Arts or Associate in Sciences degree program are generally capable of working as a junior computer programmer or programmer-trainee. Most graduates go into commercial applications work, into banks, insurance companies, small business, etc. Others go into statistical data centers, service bureaus, or local government data processing. Although most commu-

nity and junior colleges are required to perform student follow-up studies, most is done on a college-wide basis rather than for any particular curriculum. Therefore little has been documented on graduates of two year programs in computer education.

Most two-year educators feel that placement of graduates has been good. Many graduates have, within three years, moved up the promotion ladder into a variety of jobs, ranging from senior programmer, systems programmer, to management in a small company. Often, within five years of graduation, almost a third have not only been promoted on the job, but have acquired a baccalaureate degree, generally by continuing to an upper division institution on a part-time basis. Employment practices are, however, often criticized by these community and junior college program coordinators, to the extent that many local businesses and industries require that the entry level position be that of an operator, sometimes even a peripheral equipment operator. This mismatch of training to the job is usually "corrected" once the applicant has proven valuable.

More and more baccalaureate programs are becoming available in computer education, with the number of them recently passing the number of associate level programs.[13] Many companies now show a definite preference for the four year graduate, as does the U.S. Government. Two year college graduates generally have their best opportunity, in their first job, with the small systems shop. Recently, several colleges have specifically attempted to outline a separate option within their program to fill that need.

OPPORTUNITIES AND CHALLENGES

Students interested in the computer field as a profession have opportunities at the community and junior college level to learn about the field, gain good technical skills in computer programming, and get a good general education. This initial investment is usually enough to give a talented person enough foundation to allow continuation in further learning while working in the field. Community colleges generally cost less, do not require students to move away from home while going to school, and in most cases provide a somewhat more personalized atmosphere than do the large state colleges and universities.

However, certain problems do exist. Many center around the problems arising from the lack of well-qualified faculty. It is difficult to find, and keep, faculty with good academic credentials, several years of experience doing computer work, and several years of teaching experience. With the declining enrollments forecast by professional educators, administrators are hesitant to hire new full-time faculty, and are, instead, promoting "cross-training" of those faculty in other departments whose enrollments have already declined. This can be done, and can be done well, but it requires more time than is generally given. Secondly, problems arise, and are increasing, from the lack of proper equipment, or lack of proper access to the equipment to be used, or lack of control over the types of activities to be done on the equipment. Thirdly, problems exist in the lack of commitment on the part of the institution to ensure a quality program. It is difficult for instructors teaching full-time to properly revise and maintain both a facility and a curriculum, without needed released time from course instruction. Fourthly, problems exist in the lack of understanding of the students about what the career field is, and the types of skills and talents that success in the field requires. The mystique of the "computer" has not been disspelled from entering students. Lastly, problems exist in the transition of the student from one level of schooling to another, commonly called the "articulation" problem. It occurs between secondary schools and community colleges as much as it does between community colleges and upper division institutions.

NEEDS FOR THE FUTURE

Developments in the computer field continue to occur at extremely fast rates. The boom in minicomputer and microprocessor technology is having a profound effect on business and industry. The growth in small diskette and cassette computers has led to an increased need for personnel who can serve as a combination of operator, analyst, data entry operator, and programmer. The effect of data base management inquiry systems and the creation of the position of data base administrator will change the way the conventional computer programmer performs work.

Educational institutions often react slowly to the need for change. Curricula established a decade ago are, in many instances, in need of attention in both the content area and in the physical facilities being used. Several rather urgent needs, given here in prioritized order, exist:

1. A return to the philosophy of attempting to do those things that are "academically" better, rather than doing merely those things that are the absolute minimum for retaining the program. We should be looking for the best teachers, the best in facilities and the best access to them, and strengthening of the content of the programs, both technically and otherwise. Getting this is NOT always a matter of increasing costs.
2. An emphasis on a well-qualified, kept-up-to-date faculty, given time for curriculum development as well as for teaching. There should be opportunities for "cross-trained" teachers to gain experience by working under the auspices of the college, perhaps, before being placed technically "over their heads" into a classroom. Special programs should be considered for community college instructors, perhaps on a state-wide or regional basis. Opportunity to interact with other teachers, as is possible in the ACM's Special Interest Group for Computer Science Education,[14] should be found. Access to recent computer literature should be planned, by means of journals, newsletters, etc.
3. More imaginative use of the monies spent on computer equipment. Recently decreased costs of equipment can make an older five-year lease very expensive. Alternative types of computers should be studied and con-

sidered for use in an entire curriculum, or in one course. Access to this equipment should be planned under the philosophy recommended in 1. above.

4. An increased emphasis on career education in the computer field, from the middle school level, at least. Better advising and counseling is needed in both secondary schools and community colleges. Encourage professional societies to revise career leaflets as they become dated, and urge students and counselors alike to use career books presently available.[15,16]
5. Availability of a "certification" examination suitable to the requirements of the entry-level computer programmer. Such an examination would give potential employers a reference point for the evaluation of the knowledge base of the entry-level applicant. It would provide a means for continuous professional over-view of what the entry-level person should know, as continuing re-evaluation of the examination occurs from year to year. It would provide the community and junior colleges with some statistical data on the scores of their graduates for continuous evaluation of their program.

OUR ALTERNATIVES

Community colleges have experienced explosive growth over the last few decades. Programs established during that period are in need of attention. Those that get it, and a rebirth of commitment, should survive, and will perform a service to society in the preparation of well-qualified persons for the use of computers in the future. Those programs that continue without changing to meet industry requirements will, in due time, slowly fade away. The needs will be met somewhere else, perhaps by a program at the four-year level, in the state colleges or universities. The decision, and the challenge, is ours.

REFERENCES

1. Cohen, Arthur M., *A Constant Variable: New Perspectives on the Community College,* Jossey-Bass, Inc., 1971.
2. Johnson, B. Lamar, *Islands of Innovation Expanding: Changes in the Community College,* Glencoe Press, 1969.
3. *Electronic Data Processing—1, A Suggested Two-Year Post High School Curriculum for Computer Programmers and Business Applications Analysts,* U.S. Dept. of HEW, Office of Education, FS 5.280.80024, U.S. Government Printing Office, Washington, D.C., 1963.
4. *Electronic Data Processing in Engineering, Science, and Business,* U.S. Department of HEW, Office of Education, FS5.280.80030, U.S. Government Printing Office, Washington, D.C., 1964.
5. *Scientific Data Processing Technology, A Suggested Two-Year Post High School Curriculum,* U.S. Dept. of HEW, Office of Education, FS5.280.80068, U.S. Government Printing Office, Washington, D.C., 1970.
6. *Data Processing Technology, A Suggested Two-Year Post High School Curriculum,* U.S. Dept. of HEW, Office of Education, Stock Number 1780-01240, U.S. Government Printing Office, 1973.
7. Brightman, R. W., Editor, "The Computer and the Junior College: Curriculum," American Association of Community and Junior Colleges, One DuPont Circle, N.W., Washington, D.C., 20036, 1970.
8. Hill, James and Roy Sedrel, "The Computer and the Junior College," American Association of Community and Junior Colleges, One DuPont Circle, N.W., Washington, D.C., 20036, 1969.
9. Correspondence from American Association of Community and Junior Colleges, December 1975.
10. Little, Joyce Currie, Richard H. Austing, Gerald L. Engel, Harice Seeds, and John Maniotes, "Curriculum Recommendations and Guidelines for the Community and Junior College Career Program in Computer Programming, A Working Paper of the Association for Computing Machinery Committee on Curriculum in Computer Sciences, Sub-committee on Community and Junior College Curriculum," *SIGCSE Bulletin,* June 1977.
11. "Curriculum '68 Recommendations for Academic Programs in Computer Science, A Report of the ACM Curriculum Committee on Computer Science," *Communications of the ACM,* Volume 11, Number 3, March 1968.
12. Cougar, J. Daniel, Editor, "Curriculum Recommendations for Undergraduate Programs in Information Systems, A Report of the ACM Curriculum Committee on Computer Education for Management," *Communications of the ACM,* Volume 16, Number 12, December 1973.
13. Hamblen, John W., *Computer Manpower—Supply and Demand—by States,* Information Systems Consultants, RR 1, Box 256 A, St. James, Missouri, 65559, 1975.
14. For further information, write to Association for Computing Machinery, 1133 Avenue of the Americas, New York, N.Y. 10036.
15. Perhaps the best of these was *Facts on Computer Careers,* American Federation of Information Processing Societies, Inc., 210 Summit Avenue, Montvale, New Jersey 07645.
16. Maniotes, John and James S. Quasney, *Computer Careers: Planning, Prerequisites,* Potential, Hayden Book Company, 1974.

Business/computer science curricula—A survey

by KATHRYN L. SCHENK and JAMES R. PINKERT
California State University, Chico
Chico, California

INTRODUCTION

The use of computers in business is increasing dramatically, both in the diversity of applications and in the number of smaller firms now employing them. Hence, one would expect a corresponding increase in the number and extent of combined business/computer science programs in undergraduate college and university curricula. In an effort to explore this increase, and to provide one resource for their work on a combined program at California State University, Chico, the authors did a survey of such programs in the United States. This paper is a report on the results of that survey.

METHODOLOGY

The survey was conducted during the 1976-77 academic year. Program directories, departmental brochures, catalogs, professional journals, mail inquiries, and personal discussions were used to compile an initial "candidate" list of schools and their programs.

Sub-minimal crossovers were eliminated during this compilation; "minimal" was arbitrarily taken to mean "a combined program in which the degree area required at least 75 percent of the units normally required for a minor in the other area;" e.g., if a typical minor was 20 units, then a business program which required fewer than 15 units of computer-related courses was considered sub-minimal.

As indicated above, this cutoff was arbitrary. It was unfortunately necessitated after initial work indicated a very large candidate list, but one in which a significant number of supposedly combined programs required only two or three courses in the non-degree area.

It is interesting to note that during the search process the perspective of the survey changed. It was originally intended to have two aspects: computer science majors with business options and business majors with computer science options. However, the authors were able to identify very few of the former; hence, the focus is on the latter.

After compilation of the candidate list, work was begun on an in-depth analysis of each program and the courses in it. At this stage some programs were unfortunately dropped from consideration because precise requirements and/or descriptions for course numbers were not available in time. The final sample consisted of 56 schools offering a business degree with a minimal option in computer science.

Results were tabulated according to a course list derived from the union of all the samples' courses. Since emphasis here is mainly on content rather than quantified amounts, it was decided not to distinguish between quarters, semesters, 4-1-4, etc. Similarly, only two status categories were used: required and elective. Required courses are those explicitly and individually specified. Elective courses comprise all others, including such listings as "select two of the following six," etc.

The percentage of schools having each course required, elective, and either required or elective is shown in Appendix I. Appendix II is a selection of those courses most often required. Finally, Appendix III shows several of the more comprehensive programs.

COMMENTS

It must be stressed that the fundamental orientation of this survey has been the general composition of computer science options in business curricula. Ideally, of course, an advisor from every school would have been interviewed with regard to the program in his/her department. Limited resources, time constraints, and the number of programs involved made this impossible. Therefore, although every possible effort was expended to insure accuracy and completeness, no claims can be made as to the inclusion of all programs or the exact specification of programs included.

The survey results have proven very useful for the authors' purposes. It should be of similar use for others designing combined programs.

In closing, two sidelights were of interest to the authors. First, the large number of courses coupled with the small percentage of schools requiring many of these courses seems to indicate considerable diversity of opinion as to which topics should be part of such a program. Second, the topic "security" was automatically entered at the start of the tabulations. Given the concern with computer crimes and abuses, it was surprising to find that none of the programs included a course in security as either a requirement or elective.

APPENDIX I—REQUIRED AND ELECTIVE PERCENTAGES

	REQUIRED	ELECTIVE	TOTAL
I. Introduction and Languages			
1. Introduction to computers	85.7	1.8	87.5
2. Flowcharting	3.6		3.6
3. Documentation	1.8		1.8
4. Basic	1.8		1.8
5. FORTRAN I	37.5	3.6	41.1
6. FORTRAN II	5.4	3.6	8.9
7. R.P.G.	8.9		8.9
8. COBOL I	51.8	7.1	58.9
9. COBOL II	10.7	5.4	16.1
10. PL/I	7.1	3.6	10.7
11. Assembly I	30.4	8.9	39.3
12. Assembly II	1.8	1.8	3.6
13. Survey of Languages	8.9	7.1	16.1
14. Programming Applications	23.2	5.4	28.6
15. Advanced Programming	5.4	1.8	7.1
II. Foundations			
1. Data Structures	19.6	5.4	25.0
2. Numerical Analysis	1.8	1.8	3.6
3. Discrete Structures	5.4		5.4
4. Information Theory	1.8		1.8
5. Automata Theory	1.8		1.8
6. Symbolic Logic	1.8		1.8
7. Analysis of Algorithms	7.1		7.1
8. Formal Languages	3.6		3.6
9. Systems Theory	1.8	1.8	3.6
III. Hardware and Operations			
1. Computer Organization	5.4	3.6	8.9
2. Hardware Systems	3.6	3.6	7.1
3. I/O Devices	1.8		1.8
4. Systems Programming	12.5	1.8	14.3
5. Compilers	1.8		1.8
6. Operating Systems I	8.9	3.6	12.5
7. Operating Systems II		3.6	3.6
8. Real Time Systems	5.4	3.6	7.9
9. Selection of Hardware Systems	3.6		3.6
10. Use of Software Packages	1.8		1.8
11. Analogs	1.8		1.8
12. Minicomputers	1.8		1.8
13. Management of Data Processing Systems	8.9	1.8	10.7
14. Person/Machine Interaction		1.8	1.8
15. Security			
IV. Information Systems			
1. Information Systems I	17.9	1.8	19.6
2. Information Systems II	3.6		3.6
3. Accounting Systems	5.4	1.8	7.1
4. Management Information Systems	21.4	3.6	25.0
5. Systems Analysis	46.4	5.4	51.8
6. Systems Design	41.1	1.8	42.9
7. Files and Data Management	16.1	3.6	19.6
8. Information Retrieval	7.1	1.8	8.9
9. Database Management Systems	8.9	1.8	10.7
10. Case Studies		1.8	1.8
V. Probability and Statistics			
1. Probability	14.3	1.8	16.1
2. Statistics I	28.6	3.6	32.1

3. Statistics II	14.3	3.6	17.9
4. Sampling	1.8	1.8	3.6
5. Regression	3.6		3.6
6. Non-Parametric Statistics	3.6	3.6	7.1
7. Multi-Variate Analysis	1.8	1.8	3.6
8. Statistical Experiments		1.8	1.8
9. Reliability Statistics	1.8	1.8	3.6
10. Queing Theory		1.8	1.8
11. Decision Theory	10.7		10.7
12. Forecasting	1.8	3.6	5.4
VI. Operations Research			
1. Operations Research I	23.2	5.4	28.6
2. Operations Research II	5.4	8.9	14.3
3. Optimization		1.8	1.8
4. Linear Programming		1.8	1.8
5. Non-Linear Programming		1.8	1.8
6. Dynamic Programming		1.8	1.8
7. Integer Programming		1.8	1.8
VII. Simulation and Modeling			
1. Simulation	21.4	19.6	41.1
2. Simulation Languages		1.8	1.8
3. Modeling	5.4	1.8	7.1
4. Deterministic Modeling	3.6		3.6
5. Stochastic Modeling	3.6	1.8	5.4
6. Continuous Systems		1.8	1.8
7. Discrete Systems	1.8	1.8	3.6
VIII. Computer Science Topics			
1. Symbol Manipulation		1.8	1.8
2. Natural Language Processing	1.8		1.8
3. Computer Assisted Instruction	1.8		1.8
4. Artificial Intelligence	3.6	1.8	5.4
5. Pattern Recognition	1.8		1.8

APPENDIX II—TOP 18 REQUIRED COURSES

Name	% Requiring
Introduction to Computers	85.7
COBOL I	51.8
Systems Analysis	46.4
Systems Design	41.1
FORTRAN I	37.5
Assembly I	30.4
Statistics I	28.6
Programming Applications	23.2
Management Information Systems	21.4
Simulation	21.4
Data Structures	19.6
Information Systems I	17.9
Files and Data Management	16.1
Probability	14.3
Statistics II	14.3
Systems Programming	12.5
COBOL II	10.7
Decision Theory	10.7

APPENDIX III—SOME SAMPLE PROGRAMS

I. Required:
- Introduction to Computers
- FORTRAN I and II
- COBOL I and II
- Assembly I
- Data Structures
- Analysis of Algorithms
- Management Information Systems
- Systems Analysis
- Systems Design
- Files and Data Management

II. Required:
- Introduction to Computers
- FORTRAN I
- COBOL I
- Assembly I
- Data Structures
- Management Information Systems
- Statistics I and II

Sampling
Regression
Decision Theory
Operations Research I

III. Required:
Introduction to Computers
Survey of Languages
Programming Applications
Data Structures
Discrete Structures
Symbolic Logic
Computer Organization
Compilers
Probability
Statistics I and II
Forecasting

IV. Required:
Introduction to Computers
COBOL I
Assembly I
Data Structures
Discrete Structures
Information Theory
Automata Theory
Systems Programming
Information Retrieval
Natural Language Processing
Computer Assisted Instruction
Artificial Intelligence
Pattern Recognition

V. Required:
Introduction to Computers
FORTRAN I
COBOL I
Information Systems I and II
Files and Data Management
Statistics I
Simulation
Deterministic Models
Stochastic Models

VI. Required:
Introduction to Computers
Data Structures
Analysis of Algorithms
Computer Organization
Minicomputers
Systems Analysis
Systems Design
Statistics I and II
Simulation
Deterministic Models
Stochastic Models

A brief survey of computer science and engineering education

by C. V. RAMAMOORTHY
University of California
Berkeley, California

INTRODUCTION

In this paper we shall attempt an overview of some significant developments that have taken place over a span of three decades in the Computer Science and Engineering (CSE) education and to discuss some very critical issues that face the educators and the professional societies. Unlike other technologies, the computer technology has been in a state of explosive evolution. It has never been easy to define or even agree upon what should constitute an essential core of knowledge a student should obtain (learn) during a four-year period of undergraduate study of the computer technology.

With the development of the first electrical computers,[22] Harvard (1937), Bell Telephone Laboratories (1937) and the University of Pennsylvania (1946), the computer industry has grown by leaps and bounds. Computer sales *currently* amounted to more than 10 billion dollars in 1975 and it is predicted that the industry will experience a 19 percent growth in revenues by 1985, with more than two million people employed in computer related jobs. One cannot forget that we as a nation are wholly dependent on the computer. It becomes therefore, very important for us to study the role and trends of CSE education especially as it attempts to satisfy the growing need for well-trained professionals in this area.

HISTORICAL DEVELOPMENTS

It is not easy to trace the evolution of CSE education. Universities such as Harvard, MIT, Pennsylvania, and Illinois were probably among the first to include CSE courses in their regular curricula. Two types of courses were most prominent; the logic design courses primarily emphasizing the computer design techniques based on the switching theory and circuit design, and programming courses emphasizing machine and assembly language programming. While the former were generally taught by electrical engineering professors, the programming courses were taught usually by the specialists of the university's computing center under the auspices of the mathematics department. As the need for computer engineers and computer programmers increased, more and more courses were added and a distinctive pattern emerged in the administration of these courses. Electrical engineering (EE) departments naturally included computer design courses in their curriculum whereas the mathematics departments generally handled programming and numerical analysis courses.

The early computers such as Harvard Mark I, ENIAC etc., were built through the support of the U.S. Army and were intended to calculate the ballistic trajectory tables. This heralded the increasing use of computers in scientific computations, which in turn placed demands for more college graduates with numerical analysis and computer programming backgrounds. Consequently, mathematics departments initiated scientific programming courses in their curricula, and expanded their numerical analysis programs. Computer science (CS) departments began to appear in the middle sixties generally as off-shoots of mathematics departments in the Colleges of Arts and Science. Stanford University's Department of Computer Science under the late Prof. Forsythe was among the earliest. At present, the CSE education in major universities and colleges are administered mainly either by separate Computer Science and Electrical Engineering Departments, or by a combined Electrical Engineering and Computer Science Department. The former pattern can be seen in universities like Illinois at Urbana, Stanford, and Texas at Austin. The latter plan of a combined department is credited to be originated at University of California at Berkeley and at MIT,[19] and is followed at universities like Columbia, Northwestern, etc. The hardware design oriented courses are taught very naturally by the EE departments whereas the CS departments emphasize theory and programming. There are, of course, exceptions to all the above trends. Recently a few EE departments identified computer engineering as one of their major components and call themselves Electrical and Computer Engineering Departments, e.g., University of Michigan, Ann Arbor and University of Wisconsin, Madison. Also, CS programs are sometimes included in Departments of Mathematics, or Statistics or Industrial Engineering.

We have used the terms computer science (CS) and computer engineering (CE) without defining them. Perhaps the distinction between them can best be made by taking a closer look at each. The computer scientist is interested in

the theory and science of computation and programming. Thus, areas such as automata theory, formal theory of languages, complexity theory, numerical analysis and the mathematical foundations of algorithms, and data structures (thus, the science of programming a la Wirth) cover the discipline of computer science. The computer engineer is interested in the specification, design, implementation, and utilization (operation) of data processing systems including both hardware and software. Thus, computer engineering can be defined as that branch of engineering concerned with the organization, design, and utilization of digital processing systems as general purpose or specialized computers or as components of larger systems.[2] Thus the computer engineer (including the software engineer) uses the principles of computer science and/or electrical engineering in specifying, designing, implementing, and utilizing computer systems for specific applications.

CURRICULUM DEVELOPMENT

Three basic milestones can be identified in the curriculum development of the CSE education. They are (1) ACM Curriculum 68, (2) COSINE Committee recommendations, and (3) Computer Society's Model Curriculum Subcommittee (CSMCS) recommendations.

(1) *ACM Curriculum 68*—The ACM's Curriculum Committee on Computer Science (C^3S) published a set of recommendations called the Curriculum 68,[1] in March 1968. This was a comprehensive set of proposals for four-year undergraduate programs and some graduate programs. It also included description of contents of supporting courses in mathematics. In a previous set of recommendations, the committee devoted considerable attention to the justification and description of computer science as a discipline. Curriculum 68 was adapted by several universities as a model for their computer science programs. Its major effort had been to help standardize the undergraduate and graduate programs in many CS departments. The Curriculum 68 provided a clear description of course contents together with pertinent technical references. It also specified a sequence of core or essential courses that should be taught in a four-year undergraduate curriculum. Aiming primarily at programming and mathematically oriented computer science areas, it shied away from any recommendations for engineering oriented computer education.

(2) *COSINE Committee*—In the middle sixties, the COSINE Committee of the National Academy of Engineering studied the computer options in Electrical Engineering Departments. In a series of reports, they discussed various aspects of computer education in electrical engineering, including courses in computer organization,[4] digital subsystems,[3,5] digital systems laboratories,[6] and minicomputers.[8] An important COSINE report known as the Coates Committee's report[2] defined the computer option in electrical engineering which set forth a basis for the development of many future computer engineering programs. Also, the COSINE Committee's report on operating systems[7] provided the first comprehensive course description in that important area. Judging from the number of electrical engineering departments having computer science and engineering options patterned after COSINE Committee reports, it can be said that COSINE has been quite successful. The COSINE Committee's workshops and reports helped to initiate several faculty in EE department into computer engineering and helped make CE a major component of EE.

(3) *Computer Society's Efforts*—The Model Curricula Subcommittee of the Computer Society's Education Committee started efforts in 1974 to develop a four-year curriculum in CSE. "Development of model curricula that mesh computer science and engineering has, over the past decade, been a tar pit and many great and powerful beasts have thrashed violently in it."[10] The Subcommittee divided the CSE program into four subject areas and have provided course definitions and contents in each. The subject areas are digital logic, computer organization and architecture, operating systems and software engineering, and theory of computing. It is evident that these subject areas are quite broad and no single four-year curriculum could treat them adequately. Hence, the Subcommittee has proposed model curricula which are formed with elements considered essential in each of these subject areas. A number of options in the proposed curricula would provide some specialization. Increased specialization is provided at the graduate level.

In a way, the Computer Society's Model Curriculum objectives were different from those of ACM's Curriculum 68, the major difference being that the former recognized the need for both computer science and computer engineering in a single curriculum. In that sense, the Computer Society's Model Curriculum recommendations "define curricula that are interdisciplinary in nature with emphasis placed both on computer engineering and computer science." This was also the first effort to bridge the gap between computer science and computer engineering and hardware and software. To date, a number of reports have been published by the Model Curricula Subcommittee which discuss some critical issues and rationale for their recommendations.[10,11] Another significant contribution of the Computer Society is the report on Computer Architecture Curriculum.[12] In this study several well-known educators, engineers, and scientists have identified the functions and the role of the computer architect and then proposed a broad based curriculum consisting of both computer science and computer engineering subjects to educate future computer architects.

In effect, the recommended curricula stress good basic scientific and engineering education which provide the student the potential to continue education all through the professional career to assimilate and use the advances of computer technology.

DISE committee's work

Digital Systems Education Committee (DISE) was formed in 1974 with the support of National Science Foundation with the object of providing quality educational materials in the general area of digital systems. The purposes of the

project and the goals are [CAIN]: (1) to develop and/or coordinate the development of educational/instructional materials in the digital systems area; (2) to achieve widespread dissemination and use of developed materials; and (3) to encourage, develop and facilitate mutual cooperation and information transfer between the academic and industrial sectors. The ultimate objective of the DISE project was to help the educators to keep the courses and curricula current and in step with the rapid technological and theoretical developments. The committee solicited and supported proposals for developing educational materials such as textbooks, lecture material, videotapes, and self-paced instruction books in various digital system topics. The Committee also provided support for review, evaluation, and testing of these materials. A repository was established under the direction of Prof. C. L. Coates of Purdue University to collect unpublished digital systems educational material and to disseminate information regarding the items in the collection. DISE Committee has closely collaborated with the Computer Society's Model Curriculum Subcommittee using the latter's curriculum recommendations for developing their educational materials. They sponsored workshops on Microprocessors and Education which brought the industry and university engineers together to exchange latest technological information.

CONCLUSION

An extended and more comprehensive evaluation of CSE education will appear in the September issue of the IEEE Proceedings.

ACKNOWLEDGMENT

The author is grateful to Prof. C. L. Coates, Prof. Edwin Jones, Dr. M. Mulder, Prof. D. Rine, Prof. Martha Sloan, Prof. Della Bonnette, and Prof. T. Walker for discussions and suggestions.

REFERENCES

1. ACM CCCS, "Curriculum 68," Comput. Assoc. Comput. Mach., Mar. 1968.
2. Coates, C. L., et al., "An Undergraduate Computer Engineering Option for Electrical Engineering," *Proc. IEEE,* Vol. 59, June 1971, pp. 854-860.
3. COSINE Committee, "Some Specifications for a Computer-Oriented First Course in Electrical Engineering," Commission on Eng. Education, Washington, DC, Sept. 1968.
4. ———, "An Undergraduate Electrical Engineering Course on Computer Organization," Commission on Eng. Education, Washington, DC, Oct. 1968.
5. ———, "Some Specifications for an Undergraduate Course in Digital Subsystems," Nat. Academy of Engineering, Washington, DC, Nov. 1968.
6. ———, "Digital Systems Laboratory Courses and Laboratory Developments," Nat. Academy of Engineering, Washington, DC, Mar. 1971.
7. ———, "An Undergraduate Course on Operating Systems Principles," Nat. Academy of Engineering, Washington, DC, Apr. 1972.
8. ———, "Minicomputers in the Digital Laboratory Program," Nat. Academy of Engineering, Washington, DC, Apr. 1972.
9. Engel, G. L. and B. H. Barnes, "The Revisions of Curriculum 68: Background and Initial Development," in *Proc. IFIP Second World Conf. on Computers in Education,* American Elsevier, New York, NY, 1975, pp. 263-272.
10. Mulder, M., "Model Curricula for Four-Year Computer Science and Engineering Programs: Bridging the Tar Pit," *Computer,* Dec. 1975.
11. ———, "Model Curricula for Four-Year Computer Science and Engineering Program—A Progress report," *COMPCON Proc.,* Feb. 1976.
12. Rossman, G. E., et al., "A Course of Study in Computer Hardware Architecture," *Computer,* Dec. 1975.
13. Sloan, M. E., "Survey of Electrical Engineering and Computer Science Departments in the U.S.," *Computer,* Dec. 1975.
14. Snyder, J., "Certification Issues," *COMPCON,* Feb. 1975.
15. U.S. Dep. of Labor, *Occupational Outlook Handbook,* 1974-1975 Edition Bulletin 1785, Bureau of Labor Statistics.
16. U.S. Dep. of Labor, "Ph.D. Manpower: Employment Demand and Supply 1972-1985," *Bulletin 1860,* Bureau of Labor Statistics.
17. U.S. Dep. of Labor, "Technological Change and Manpower Trends in Five Industries," Bureau of Labor Statistics, 1975.
18. Walker, T. M., "Computer Science Curricula Survey," *SIGCSE Bulletin,* Vol. 5, Dec. 1973, pp. 19-28.
19. Zadeh, L. A., "Computer Science as a Discipline," *J. Eng. Education,* Vol. 58, Apr. 1968, pp. 913-916.
20. Freeman, P., A. I. Wasserman, and R. E. Fairley, "Essential Elements of Software Engineering Education," in *Proc. 2nd Int. Conf. on Software Engineering,* Oct. 1976.
21. Crockett, D., "Private Communications on Guidelines for Accreditation of Computer Engineering Programs," May 1976.
22. Huskey, H. D. "Chronology of Computing Devices," *IEEE Transactions on Computers,* Dec. 76. pp. 1190-1199.

A panel session—Accreditation of computer-oriented academic programs

SESSION CHAIRMAN—GERALD E. WAGNER
California State Polytechnic University, Pomona

Panel Members

Thomas H. Athey—California State Polytechnic University
Sonya L. (Sam) Anderson—T.H. Garner Company
Donald B. Medley—Moorpark College
Eugene B. Smith—U.S. Department of Agriculture

PANEL OVERVIEW—Gerald E. Wagner

The term "accreditation" is defined by the Standard College Dictionary as, "The granting of approved status to an academic institution by an accrediting body after an examination of its courses, standards, etc." Traditionally different levels of accreditation have been available. For example, individual degree programs (majors) can be accredited, total programs can be accredited such as is provided by the American Assembly of Collegiate Schools of Business (AACSB), and entire university programs can be accredited.

This panel will limit itself to the topic of accreditation as it applies to a single academic program. The actual name of the computer degree program is insignificant for the purposes of this presentation and all computer-oriented degree programs are included whether they be called "Computer Science," "Business Data Processing," "Information Systems," "Information Science," or something else. The entire subject or accreditation will be reviewed—both pro and con—as it relates to these programs.

Any decision to accredit or not to accredit academic computer degree programs could have a serious impact on many different groups within our society. The positive and negative aspects of accreditation will be discussed as it relates to: (1) students (both Community College and University), (2) degree-granting institutions, and (3) future employers of the graduates of the degree programs.

Many different professional organizations have attempted to define curricular patterns, but as of this date, no attempt has been made to provide a method for evaluating academic programs. These past efforts have not been wasted though and any efforts to develop accreditation "standards" could and should utilize much of knowledge gained through experiences of these other groups. Some of the relevant efforts in this area will be discussed as they relate to the accreditation process.

Further insight may be gained regarding the accreditation process by reviewing the history of other accreditation groups. In particular, the American Assembly of Collegiate Schools of Business and the Engineers Council for Professional Development will be discussed for that purpose.

Although accreditation has its inherent problems, the benefits to be gained appear to outweigh any of the potential problems or disadvantages that may result. The panel members recommend that immediate steps be taken to develop a workable accreditation process. The first step to be taken would be to establish a committee with representatives from business, professional associations and academic institutions. Furthermore, it is recommended that any efforts to develop accreditation standards must recognize the different roles that different types of computer degree programs play in our society. Any standards must be designed to encourage creativity—not stifle it—as long as the "creativity" does not detract from or change the overall objectives of the academic program.

ACCREDITATION OF COMMUNITY COLLEGE DATA PROCESSING PROGRAMS—Don B. Medley

Black's Law Dictionary defines accreditation as "To send with credentials as an envoy." The New Century Dictionary by Appleton-Century-Crofts, Inc. defines accredit as follows:

> To bring into credit; invest with credit or authority; also to send with credentials; also, to give credence to; believe; trust; also, to credit with something ascribed.

My concern is that we in higher education are sending our graduates out with credentials as an envoy and the credentials are being rejected. Considerable effort has been expended by the government and the several professional organizations in the definition of curriculum patterns for programs at all levels of higher education but little has been done to accredit or validate the programs after they are implemented. Probably the most active organization to work on curriculum development and design has been the Association for Computing Machinery (ACM). A curriculum working paper containing recommendations and guidelines

for a community and junior college career program in computer programming was published in the Special Interest Group on Computer Science Education (SIGCSE) bulletin in June, 1977.

The paper addresses the courses required in such a program and the detail content of each course is defined. The instructional resources, equipment and faculty needs are covered but the only comments related to accreditation deal with articulation. While this developmental work and the followup effort defined in the articulation section are extremely important and must be continued and expanded, there is also a pressing need for a formal evaluation program that will accredit the offerings of particular institutions.

An evaluation system for community college occupational program (COPES) has existed in California since 1971. COPES is a system for voluntary evaluation of occupational programs administered through the Chancellor's Office, California Community Colleges in Sacramento, California. Among the top strengths of the COPES evaluation as rated by the COPES teams are:

- Qualifications of instructional staff
- Occupational experience of instructors
- Quality of occupational instruction
- Administration's commitment to occupational education
- Adequacy and availability of instructional materials
- Updating of instructional content and method
- Utilization of instructional facilities and equipment

Another case for accreditation standards is developing with the expansion of the programs offering college credit for "Life Learning and Experience". Perhaps the professional associations could work with the College Entrance Examination Board and have the information systems included in their long-range examination of American education called Future Directions for a Learning Society.

ACCREDITATION: PROBLEMS AND PERSPECTIVE—Eugene B. Smith

Computer Science education has evolved over the last 25 years to a significant element of many academic programs. The level of training varies from introductory material for all students to the research oriented PhD degree. The scope of the material covered and the location of the instruction within the Univeristy vary widely. The level of maturity of this field dealing with computers and their use has reached a point where there should be a concerted effort to initiate some predefined measure of quality control over the instruction being provided. Accreditation is an accepted vehicle for identifying and maintaining a specified level of quality for academic programs.

If one were to contemplate advocating accreditation for computer related educational programs, it is important to understand what the term accreditation implies. This paper will provide a discussion of the functions of accreditation, the types of accreditation, and the accreditation procedure.

Educational institutions in the United States are not controlled by a single Federal ministry. Accrediting bodies include governmental agencies, regional associations, and programmatic associations. Both institutions and program areas within institutions may receive various types of accreditation. Some consideration must be given to the levels of the institution which would be included in any accreditation effort.

A brief discussion of other accreditation groups should provide some insight into the approach which could be taken in implementing this type of effort. References to the history of the American Assembly of Collegiate Schools of Business (AACSB) and The Engineers Council for Professional Development will be provided.

A better understanding of the accreditation process and identification of the players who must participate in such an activity will provide an atmosphere within which potential problem areas can be identified. Considering the state and scope of computer oriented education, now is the time to begin an attempt to develop a workable accreditation process. A reasonable first step would be to establish a qualified task force with the dedication required to initiate an accreditation effort. A coalition of business and academic representatives, with the support of the professional associations should be formed.

ACCREDITATION—A UNIVERSITY PERSPECTIVE—Thomas H. Athey

Universities have a major part to play in the training of individuals who desire to become EDP professionals. The significance of this role will increase in the near future as user organizations progress to more sophisticated and encompassing computer-based information systems. These types of systems will require individuals who have sufficient skills at the entry level to make a meaningful contribution to the organization plus a broad enough perspective to function effectively in society. Universities are uniquely qualified to assume this responsibility.

There are many groups who should be vitally interested in the training process performed by the university. These stakeholders include employers, associated departments within the universities, junior colleges and high schools, and students. These stakeholders need reliable evidence concerning the excellence of curriculum and the resultant competency of the graduates who desire to apply these learned EDP skills.

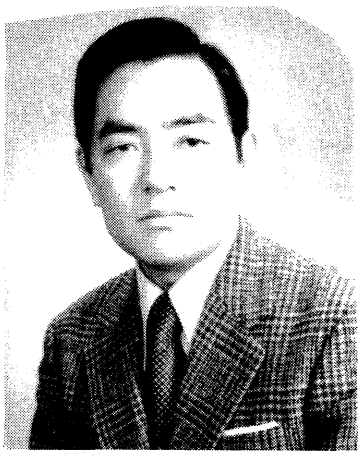
One of the ways of developing indicators of quality and competency is through the accreditation process. In the past, this has been attempted by several professional organizations within the EDP field. The initial work has been in establishing the resources utilized in the various programs to include measurement of the type of equipment available, the number of faculty, etc. The next stage in development

has been in curriculum building. Establishing the types of courses and patterns that seem to be most appropriate for acquiring particular EDP skills and perspectives. A most ambitious approach has been put forth by the Accreditation Research Committee of AACSB. They are concentrating on a value added method by which they propose to measure competence levels of knowledge, skills, aptitudes, attitudes, and personal characteristics of individuals who complete various programs.

While the major advantage of accreditation is the resulting verification process, there are several other benefits which could accrue from a well designed plan. These would include (1) development of more meaningful and effective articulation agreements between junior colleges and universities, (2) explicit recognition of the highly different orientations required of curriculum aimed at different segments of the EDP professional market, and (3) the development of aptitude tests which are much more inclusive and valid than the present programmer's aptitude tests.

The accreditation process is not without its potential disadvantages. These include (1) a strong tendency to insure uniformity of programs, which could result in stifling innovation, (2) the possibility exists for universities to become too vocationally oriented and forget that their mission is education, and (3) if any type of mass testing is required, not being able to constantly update the testing procedure to reflect the needs of the rapidly changing field.

Overall the case for an accreditation process seems very strong. However, to be successfully implemented the accreditation body must include representatives from each of the major stakeholders in the training process and useful measures of competency must be determined.



Area Director:
Hideo Aiso
Keio University
Yokohama, Japan

Recent progress in Japan

It is hoped that the area entitled "Recent Progress in Japan" will provide the audience with a better understanding of computer technology in Japan. The primary objective is to overview the present status and the future prospects of both semiconductor and computer technologies from viewpoints of recent activities in the field of research and development, and to present some main research efforts on the development of computers and their application systems suitable for Japanese society.

The first session consisting of two papers is devoted to presenting the state of semiconductor technology in Japan. Semiconductor technology is expected to be responsible for the next major evolution, not only in the computer industry but also in the electronics industry in general. The first paper includes a review of the recent advances in electron optical studies relating to electron beam pattern generation with sub-micron resolution using the variable shaped beam technique. The direct-writing electron beam technology capable of sub-micron geometry is considered to be one of the most important steps towards the achievement of advanced VLSI ICs for next-generation computers. The second paper presents research and development activities in both academic laboratories and semiconductor industries in Japan. It also explores new semiconductor devices offering excellent high-speed switching characteristics with low power dissipation, which are expected to be used for realizing future computers.

The second session is intended to overview the development of computers and technical progress in their application systems in three separate papers. The first paper describes, after a short historical background, the technological characteristics of newly developed computer series with emphasis on their architectures and LSI implementations. The research and development projects of both VLSI and PIPS (Pattern Information Processing System), which are organized by computer industries, academic laboratories and the Government are taken up briefly. The second paper discusses recent features and future prospects of remote data processing in Japan from the viewpoints of various aspects of technology, such as development background, computer network architecture or telecommunication. The final paper discusses the technological trends in real-time application systems through describing the design concept of a very reliable operation control system for the high-speed railway system called the "Shinkansen."

The third session consists of three papers concerned with recent outstanding research efforts on the development and application of relatively small computers.

The first paper gives a detailed description of a very high-performance NMOS/SOS microprocessor developed as part of the PIPS project. This microprocessor contains approximately 7,000 gates in a single chip with 80 pins, and executes most microinstructions at the speed of 200 ns. The technology applied to the realization of this microprocessor speaks well of the potential power of Japanese semiconductor technology today. The second paper describes a very interesting application of computers, namely a comprehensive automobile control system giving the driver indications of the best routes to take, as well as other information such as specific traffic regulations. Moreover, it contributes to economical use of gasoline and reduction of air pollution.

The rapid progress of LSI technology and minicomputers has great potential to build very reliable computing systems using distributed processor systems concept. The final paper describes how distributed computers can be effectively applied to advanced process industries, where control systems, that are more reliable than the ones in use now, are needed. This paper discusses the requirements, design considerations, advantages and possibilities of these systems from the viewpoints of both hardware and software developments.

Electron beam lithography for advanced LSI fabrication

by EIICHI GOTO

University of Tokyo and The Institute of Physical and Chemical Research
Wako-shi, Saitama, Japan

and

TAKASHI SOMA, MASANORI IDESAWA and TATEAKI SASAKI

The Institute of Physical and Chemical Research
Wako-shi, Saitama, Japan

INTRODUCTION

The current technology used in IC (Integrated Circuit) and LSI (Large Scale Integration) fabrication is based on optical lithography or the art of photocopying. The light wave length sets an ultimate and absolute limit on the resolution of the optical method at about one micron ($\mu m = 10^{-6}$ meter) and the current LSI technology has almost reached this limit.

In microscopy, a drastic improvement of resolution was achieved in the 1930's by switching from glass (light) optics to electron optics. The similar is taking place in lithography for LSI fabrication. X-ray lithography and electron beam lithography are the two promising methods for improving the resolution by at least one order of magnitude over optical lithography. The characteristic features of these three lithographic methods are summarized in Table I.

In Table I, "Mask in Contact" means that a mask, with a master pattern inscribed on it, is placed in close proximity to the target (silicon wafer covered with photo-resist) so as to selectively expose the target to the incident beam. The finite and rather short life of the masks in contact is the major disadvantage of this scheme.

In Table I, "Mask Projection" means that an image forming projection system (actually a lens) is placed between the mask and the target so that the mask does not have to be in contact with the target. Since there is no material suitable for making an X-ray projection lens, this scheme is not applicable to X-ray.

In Table I, "Direct Pattern Generation" means a feature in which the incident beam is maneuvered so as to generate the pattern directly on the target without using a mask. While this feature can be implemented by suitably deflecting an electron beam at high speed without any fundamental difficulties, this is not the case for light and X-rays. Note that the sub-micron pattern generation feature is also needed for making the masks themselves. Therefore, "electron beam pattern generation" is the key and the very heart of advanced LSI technology which requires sub-optical or sub-micron resolutions.

Moreover, because of flexibility and many other practical reasons, electron beam pattern generation is considered advantageous for making masks to be used with optical lithography in many cases.

The development of practical electron beam pattern generation schemes is reviewed in a later section. The recent developments in electron optics, i.e., the theoretical foundation of the image-forming processes of electron beams, are reviewed and their implications to the practical systems are discussed. Fly's eye optics which may find some use in the future is also discussed.

ELECTRON BEAM PATTERN GENERATION SCHEMES

Figure 1 shows three practical electron beam pattern generation schemes, and their developments are summarized in Table II.

In Figure 1, "Spot Scanning" means that an electron beam, with a small circular cross-section (spot), typically of 0.1 micron in diameter, is deflected electrically (by applying magnetic deflection field or electrostatic deflection field or both) and the pattern is generated by scanning the beam across the area to be exposed to the beam. In other words, the pattern is generated similarly to the pictures on a television screen, except for much increased resolution. So far as scanning and the resolution are concerned, this scheme is similar to SEM (Scanning Electron Microscope). Actually, SEM's with slight modifications were used in the early models of pattern-generating machines. For example, the first Japanese pattern-generating machine JBX-2A was developed concurrently with the first SEM, JSM-1. Using this spot scanning machine (JBX-2A), Tarui and his collaborators successfully made MOS transistors purely with electron

TABLE I.—A Comparison of Lithographic Methods

	"Mask in Contact"	"Mask Projection"	"Direct Pattern Generation"	Ultimate Resolution
Optical	0	0	x	$\gtrsim$ 1 μm
X-Ray	0	x	x	$\lesssim$0.1 μm
Electron Beam	0	0	0	$\lesssim$0.1 μm

0—Practicable. x—Impracticable.

beam pattern generation without using any masks at all in 1968 [L1].

EBES2 and VL-1 (entries 3 and 4, Table II) are also "spot scanning" machines. In EBES2 the electron beam is deflected electrically only in one direction (deflection width 0.128~0.256 mm) and the target is scanned mechanically in the other direction so as to scan and generate two dimensional patterns. In VL-1 the electron beam is deflected two dimensionally within a square scanning area of 5 mm by 5 mm.

Referring to Figure 1, "Fixed Shaped Beam" means that the cross-section of the electron beam at the target has a fixed shape, typically a square of 2.5 μm by 2.5 μm. This is achieved by projecting the image of a mask with a square hole on to the target. The pattern is generated by scanning and/or patching the area to be exposed to the beam. EL-1 is a "fixed shaped beam" machine.

In Figure 1, "Variable Shaped Beam" means that the shape of the cross-section of the beam at the target is variable, typically a rectangle x by y μm with x and y being varied from 0.1 to 25 μm. This is achieved, for example, by using two masks each having a square hole. The image of the first mask is projected on the second mask and the image of the second mask is further projected on the target. The combined action of the two masks are varied by the beam-shaping deflector placed between the two masks, and the shaped beam is directed to a selected position on the target by means of beam-positioning deflector placed between the second mask and the target. The "variable shaped beam" concept was first reported at the May 1977 EIPBT (Electron, Ion and Photon Beam Technology) Symposium by four independent groups (entries 6, 7, 8, and 9, Table II).

Figure 2 is a photograph of one of the first test patterns generated by JBX-6A, the variable shaped beam machine of the Japanese group. Entry 10 in Table II shows the result of the design and feasibility studies made by the authors' group on the variable shaped beam scheme.

Before making comparison and evaluation of the various pattern generation schemes we shall review the recent advances in electron optics.

ADVANCES IN ELECTRON OPTICS

The studies on the image-forming processes of electron beams are called electron optics. Similar to glass (light) optics, the deviations in the actual image from the ideal image are called geometrical aberrations.

The term "geometrical" means that the wave nature of the beam is being neglected. Since the wave nature or quantum mechanical effects can be neglected in the schemes shown in Table II, we shall drop the term "geometrical" hereinafter for simplicity. The aberration theory of the pure (deflection-less) image-forming process of electron beam is well established and we can learn the results from definitive text books on the subject, such as those authored by Glaser [E1] and El-Kareh [E2].

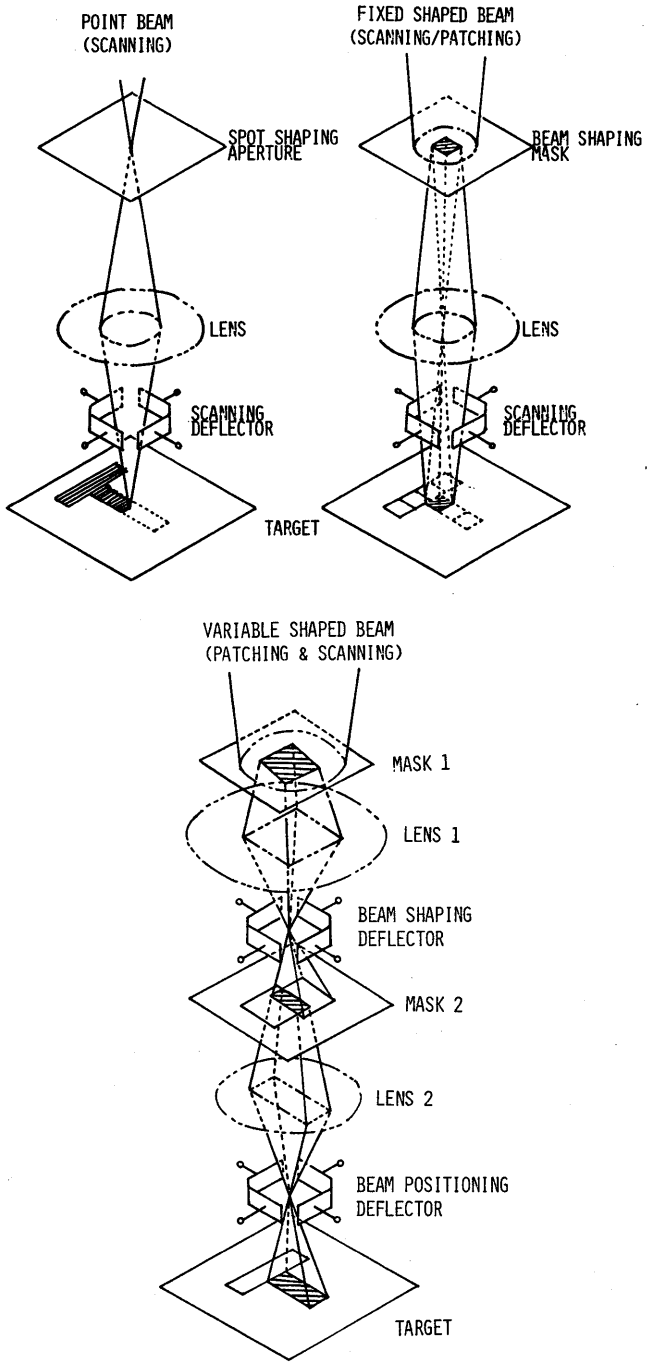


Figure 1—Electron beam pattern generation schemes

TABLE II.—Electron Beam Pattern Generators

0	Model	Beam Shape (μm)	Resolution (μm)	Total Beam Curr. (A)	Electrical Scan Width or Area	Year Reported	Developed at/by	Reference
1	JSM-1	0.1 ϕ	0.1	10 pA	1×1 mm²	1966	JEOL	[—]
2	JBX-2A	0.35 ϕ	0.35	0.01 μA	2×2 mm²	1968	JEOL&ETL	[L1]
3	EBES2	0.25-0.5 ϕ	0.25-0.5	0.1 μA	0.128-0.256 mm	1975	BTL	[L4]
4	VS-1	0.05 ϕ	0.05	1 μA	5×5 mm²	1975	IBM	[L5]
5	EL-1	2.5 □	0.4	3 μA	5×5 mm²	1976	IBM	[L7]
6	JBX-6A	0.5-25 □	0.5	1 μA	2×2 mm²	1977	IPCR&JEOL	[L9]
7	—	0.6-2 □	0.6	2 μA	4×4 mm²	1977	IBM	[L10]
8	—	0.5 □ □	0.5	0.1-0.4 μA	0.256 mm	1977	BTL	[L11]
9	—	—	—	—	—	1977	Thomson	[L12]
10	—	0.1-25 □	0.1	≥1 μA	7×7 mm²	1978	IPCR	[—]

Remarks:
1. SEM (Scanning Electron Microscope), Japan Electron Optics Laboratory Co., Ltd.
2. Spot scanning, Electrotechnical Laboratory, Japan.
3. Spot scanning, Bell Telephone Laboratories Inc., U.S.A.
4. Spot scanning, International Business Machines Corporation, U.S.A.
5. Fixed shaped beam.
6. The Institute of Physical and Chemical Research, Japan. 6,7,8,9,10: Variable shaped beam.
9. Thomson CSF, France.

However, when electrical deflection of the beam enters, the theories developed up to the 1960's are all incomplete in that the focusing field and the deflection field are spacially separated. A unified treatment of spacially superimposed focusing and deflecting fields are needed for the full development of the aberration theory and this has been accomplished only very recently.

Electron optics is a rather specialized art full of jargon of lengthy mathematical formulas and odd terminologies for non-specialists. The authors would try their best to review the advance without entering the jargon. Nevertheless, we shall need the definitions of, at least, some quantities for the sake of clarity.

Figure 3 shows the definitions of three (optical) parameters α, β and γ. We assume that there is a final electron lens (the objective lens) which projects the beam on the target. The shape of the beam on the target is called the image. The distance (specifically called the working distance in pattern-generating machines) from the objective lens to the target is denoted by L.

The parameter $\alpha=F/2=a/L$, with a being the aperture radius of the lens, is the half aperture angle at the image. The brightness of an optical lens (say of a camera) is usually measured by using an index F called the F-number. Hence, $\alpha=F/2$ means that α is the index representing the brightness of the electron lens.

The parameter $\beta=b/L$, with b being the size of the image, is an index representing the size of the electronic image relative to the (working) distance L.

The parameter $\gamma=c/L$, with c being the amount (or the length) of deflection on the target, is an index representing the amount of deflection relative to L.

The ratio $\Delta V/V$, with V being the acceleration voltage of the electron, represents the relative spread in electron energy, which may be caused by thermal effects on the cathode, variations in the accelerating voltage itself as well as by the Boersch effect to be discussed later.

The aberration theory has to do with expressing the quantity $\delta w=\delta(x+iy)$, the deviation of the actual beam point

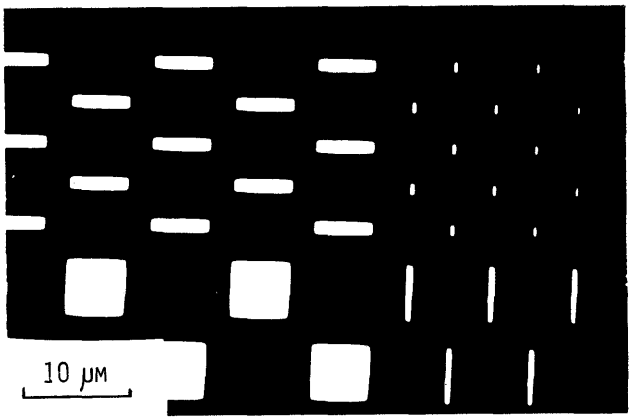


Figure 2—A test pattern generated by JBX-6A

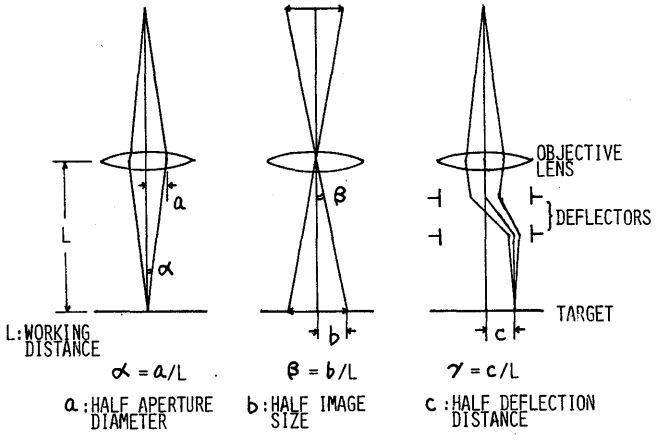


Figure 3—Three electron optical parameters: α, β and γ

TABLE III.—Advances in Electron Optics of Deflection Systems

	Field F*	D*	Content (Aberration=Ab.)		Year	Author(s)	References
1	M	M	$\exists f, f(\alpha,0,\gamma)=g(\alpha)$	Ab. Elimination	1971	Ohiwa,Goto,Ono	[E4]
2	M	M	min $\lvert f(\alpha,0,\gamma)\rvert$	Ab. Reduction	1973	Owen,Nixon	[E5]
3	M	M	$f(\alpha,0,\gamma,\Delta V/V)$	Ab. Formula	1974	Munro	[E6]
4	M	M	min $\lvert f(\alpha,0,\gamma,\Delta V/V)\rvert$	Ab. Reduction	1975	Munro	[E7]
5	M	M	min $\lvert f(\alpha,0,0,\Delta V/V)\rvert$	Ab. Reduction	1975	Pfeiffer	[E8]
6	M	M	$\exists f, f(\alpha,\beta,\gamma,\Delta V/V)=g(\alpha,\beta,0,\Delta V/V)$		1977	Goto,Soma	[E9]
7	M	M	min $\lvert f(\alpha,\beta,\gamma,\Delta V/V)\rvert$	Ab. Reduction	1977	Goto,Soma,Idesawa	[L9]
8	ME	ME	$f(\alpha,\beta,\gamma,\Delta V/V)$ Ab. Formula		1977	Soma	[E11]
9	M	E	min $\lvert f(\alpha,\beta,\gamma,\Delta V/V)\rvert$	Ab. Reduction	197x	Soma [to be published]	

F*:Focusing Field, D*:Deflection Field.
M:Magnetic, E:Electro-static, ME:Combined Magnetic and Electro-Static.

incident on the target from the ideal point, as a function of α, β, γ and $\Delta V/V$, where $x+iy$ is the two dimensional cartesian coordinate in the target plane in complex form. Namely, we have a function f such that $\delta w=f(\alpha,\beta,\gamma,\Delta V/V)$.

This function is usually expanded into a Taylor series, and each term in the series is given a specific name. For example, the terms including the energy spread index ($\Delta V/V$) are called chromatic aberrations.

The recent advances in the aberration theory and results in the design of electron optical systems with reduced aberrations are summarized in Table III.

First, note that the deflection-less ($\gamma=0$) pure image-forming case has to do with the special case $\delta w=f(\alpha,\beta,0,\Delta V/V)$, and that for "spot scanning" the point image ($\beta=0$) function $f(\alpha,0,\gamma,\Delta V/V)$ would be sufficient but the full function $f(\alpha,\beta,\gamma,\Delta V/V)$ would be needed for "fixed- and variable-shaped" beam schemes.

Ohiwa, Goto and Ono (#1 in Table III) introduced a MOL (Moving Objective Lens) concept and showed that there exists ($\exists f$) a system for point image ($\beta=0$) free of all deflection-induced aberrations, i.e., $f(\alpha,0,\gamma,0)=g(\alpha)$. Chromatic aberrations, however, were not included (assuming $\Delta V/V=0$).

Munro (#3) derived the formula for point image case ($\beta=0$) with chromatic aberrations.

Goto and Soma (#6) derived the formula for the finite image case ($\beta\neq 0$) and also theoretically proved the existence of systems free of all deflection-induced aberrations by making use of the MOL concept.

Soma (#8) derived the most general formula valid for any combination of magnetic and electro-static fields and for electrons running at relativistic speeds. The formula is some 80 pages long and the computerized formula manipulation system Reduce 2 [E12] was utilized for the deriviation. It is extremely difficult to handle such a long formula without such manipulation systems.

The other entries in Table III with "min" (for minimization) gave practical results in reducing various aberrations (there are some 20 aberrations.)

We want to emphasize that, in all the aberration reduction calculations made by the authors' group (entries 1, 6, 7 and 9 in Table II), we also required the beam to land vertically on the target. As a result of these studies, all aberrations except for two have been found to be either eliminable or negligibly small in practice. The two are the spherical aberration δw_s and the axial chromatic aberration δw_a, which have the following forms:

$$\delta w_s=S\alpha^3, \quad \delta w_a=A\alpha\Delta V/V$$

where S and A are the aberration constants. Note that these aberrations exist even in the deflection-less point image case ($\beta=\gamma=0$), i.e., $f(\alpha,0,0,\Delta V/V)=S\alpha^3+A\alpha(\Delta V/V)$. In spite of a great number of efforts trying to eliminate the spherical aberration, no practical scheme has been found yet. [E3] These two aberrations are explained pictorially in Figure 4. The spherical aberration means that electrons having larger aperture angle α are bent stronger by the lens, and the axial chromatic aberration means that slower electrons ($\Delta V/V<0$) are bent stronger by the lens.

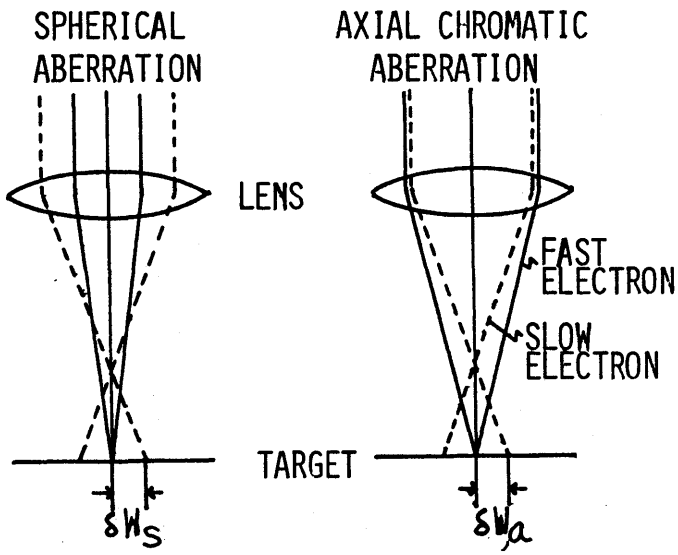


Figure 4—Spherical and axial chromatic aberration

LIMITS ON CURRENT DENSITY AND EXPOSURE SPEED

Besides geometrical aberrations there is another factor, not having its counterpart in light optics, which causes

broadening in the shape of electron beams. That is the electron-electron interaction. When the intensity of an electron beam is increased beyond a certain limit the electrons would start to repel each other and collide with others thereby broadening the shape of the beam. The theory of such broadening is not in a fully satisfactory form [B2] at present.

Among others the Boersch effect, named after the first observer of the effect, [B1], is a controversial issue. Some experimentalists have even questioned the very existence of the effect itself. [B5, B6] Boersch and some others [B6] observed broadening in electron energy ($\Delta V/V$) which is much larger than the thermal fluctuations to be expected from the cathode temperature, when electrons are passed through a spacial region of high electron density. Boersch effect would imply an increase of chromatic aberrations. In addition to the Boersch effect, the space charge causes the displacement of electron trajectories. By regarding the electron beam as a continuous and charged fluid, the displacement caused by the average (averaged over all electron trajectories) space charge effect can be easily derived, yielding the following to be called the space charge broadening formula:

$$\delta w_{sc} = KIL/(V v \alpha), \text{ (in CGS units)},$$

where α is the half aperture angle, V is the accelerating voltage and L is the path length of the electron as defined in the previous section. Further, I is the total beam current (*not the current per unit area*), v is the electron speed ($v = \sqrt{2eV/m}$), and K is a number of the order of unity, which does not depend critically upon the image size β nor on other detailed structures of the system. Thus, we may well use $K=1$ as the rule of thumb.

The experiments performed on JBX-6A (Table II, #6) indicate that axial chromatic aberration and chromatic aberration due to the Boersch effect could hardly affect the projected 0.1 micron resolution and that the observed broadening of the image agrees reasonably well with the space charge aberration δw_{sc}. Thus, the spherical aberration δw_s and the space charge aberration δw_{sc} are concluded to be the major limitations imposed on the resolution and on the beam current.

The space charge aberration $\delta w_{sc} = KIL/(V v \alpha)$ would be reduced by increasing α, the brightness of the objective lens, but this would increase the spherical aberration $\delta w_s = S\alpha^3$. Hence, there is an optimum value of α as a result of compromise. The lithographic process requires the acceleration voltage V to be around 20 K volt. Using the design figures of $L = S = 10$ cm, we arrived at the conclusion that "more than 1 micro ampere beam current would be feasible at 0.1 micro meter resolution" as indicated in Table II, #10.

The following are the implications of 1 μA beam current.

1. The lithographic sensitivity of the advanced electron resist material is better than 10^{-6} coulomb/cm^2. Hence, exposure speed of 1 cm^2/sec would be achieved (20 sec exposure time for 2 inch wafer).
2. If "spot scanning" with 1 μA beam current in a 0.1 μm spot were used, scanning speed of 10^{10} spots/sec (10 Giga-Hz) would be needed, which would be extremely difficult to engineer. For this reason alone, one would have to use "shaped beam" schemes in order to increase the scanning speed.
3. Moreover, 1 μA in a 0.1 μm spot implies 10^4 A/cm^2 in current density. A very bright cathode (e.g., field emission type) would be needed to realize such high current densities, but this would reduce the engineering freedom in the cathode design. The current density would be greatly reduced in "shaped beam" schemes.
4. From these considerations just given, "shaped beam" schemes are believed to be quite advantageous and even a necessity for attaining high exposure speed. On the other hand, "spot scanning" may be advantageous for lower speed devices because of its structural simplicity.

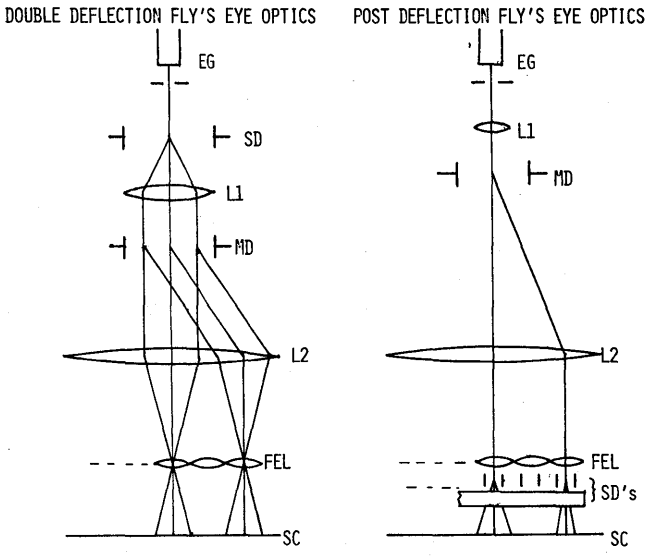


Figure 5—Fly's eye optical schemes

FLY'S EYE OPTICS FOR PURE ELECTRICAL LARGE AREA SCANNING

In all the electron beam pattern generation schemes described earlier (cf. Table II), the electrically scanned areas (typically 5×5 mm^2) are too small to cover the entire target area of silicon wafers, typically two to five inches in diameter. Hence, the target is placed on a mechanically scannable stage so as to back up the electrical scanning. The widening of the electrically scanned area is difficult because it would imply an increase in the working distance L resulting in an adverse effect on the space charge aberration $\delta w_{sc} = KIL/(V v \alpha)$.

Fly's eye optics may provide a method for scanning a large area purely electrically. Figure 5 shows two fly's eye schemes. The "post deflection" type was invented by Newberry [F1] and the "double deflection" type, by one of the present authors. [EG] [F2, F3, F6]. In Figure 5, FEL (Fly's Eye Lens) is illustrated by a matrix of lenses. In both types, the electron beam is selectively passed through a specific

lens in the FEL by the action of the main deflector MD. The beam is further deflected by sub-deflector(s) SD(s), so as to scan a small area. The difference between the two schemes consists in that while a number of small sub-deflectors are placed after the FEL in the "post deflection" scheme, a single sub-deflector placed before the FEL is sufficient in the "double deflection" scheme. Lenses L1 and L2 are inserted in the "double deflection" scheme for the explanatory purpose but they may be removed.

The "post deflection" scheme and its modifications are used in electron beam memory schemes, with sub-micron memory cells. [F4, F5, F7]. The "double deflection" scheme is used in very high precision cathode ray tubes. [F2, F3, F6].

CONCLUSION

Electron beam pattern generators with 0.1 micron resolution running at a speed greater than 1 cm^2/sec are believed to be technologically feasible, based on electron optical studies. The "variable shaped beam" scheme will be used in faster machines and the "spot scanning" scheme, in slower machines.

The impacts of such pattern generators on LSI fabrication are rather difficult to foresee precisely because of the conglomerate nature of the LSI technology. We, as specialists in electron optics, firmly believe that the electron beam can never be the bottleneck in the development of advanced LSI's with sub-micron and sub-optical patterns.

We would like to acknowledge Messrs. S. Miyauchi, K. Tanaka and T. Someya of JEOL for stimulating discussions and for providing us with the photograph of the test pattern (Figure 2).

REFERENCES

Lithography by electron beam

[L1] Tarui, Y., S. Denda, H. Baba, S. Miyauchi and K. Tanaka, "An Electron Beam Exposure System for Integrated Circuits," *Trans. IECE Japan,* 51-C,2, Feb. 1968, p. 74. (in Japanese).

[L2] Miyauchi, S., K. Tanaka and J. C. Russ, "IC Pattern Exposure by Scanning Electron Beam Apparatus," *Solid State Technol.,* July 1969, p. 43.

[L3] Herriott, D. R., D. S. Alles, R. J. Collier and J. W. Stafford, "EBES: A Practical Electron Lithographic System," *IEEE Trans.,* ED-22,7, July 1975, p. 385.

[L4] Alles, D. S., F. R. Ashley, R. J. Collier, E. A. Gere, D. R. Herriott, A. M. Johnson and M. G. R. Thomson, "A Second Generation EBES," *Suppl. IEDM Tech. Digest,* Dec. 1975, p. 1.

[L5] Speth, A. J., A. D. Wilson, A. Kern and T. H. P. Chang, "Electron-beam Lithography Using Vector-scan Technique," *J. Vac. Sci. Technol.,* 12, 1975, p. 1235.

[L6] Soma, T., M. Idesawa and E. Goto, "A New Electron Beam Exposure Scheme of Scanning Type," *IEDM Tech. Digest,* Dec. 1975, p. 20.

[L7] Yourke, H. S. and E. V. Weber, "A High-Throughput Scanning-Electron-Beam Lithography System," *IEDM Tech. Digest,* Dec. 1976, p. 431.

[L8] Mauer, J. L., H. C. Pfeiffer and W. Stickel, "Electron Optics of the Electron-Lithography System, EL-1," ibid, 1976, p. 434.

[L9] Goto, E., T. Soma and M. Idesawa, "A Design of a Variable Aperture Projection and Scanning System for Electron Beam," *14th EIP Beam Symp.,* May 1977.

[L10] Pfeiffer, H. C., "Variable Spot Shaping for Electron Beam Lithography," ibid, 1977.

[L11] Thomson, M. G. R., R. J. Collier and D. R. Herriott, "A Double Aperture Method of Producing Variably Shaped Writing Spots for Electron Lithography," ibid, 1977.

[L12] Trotel, J., "E-Beam Dynamic Shaping," ibid, 1977.

[L13] Scott, J. P., "Recent Progress on the Electron Image Projection," ibid, 1977.

[L14] Speidel, R. and M. Mayr, "Electron Beam Projection System with Photocathode," *Optik,* 48,2, 1977, p. 247.

Electron optics

[E1] Glaser, W., *Grundlagen der Elektronenoptik,* Springer, Verlag, 1952.

[E2] El-Kareh, A. B. and J. C. J. El-Kareh, "Electron Beam, Lens, and Optics," Vol. 1, and 2, Academic Press, 1970.

[E3] Septier, A., "The Struggle to Overcome Spherical Aberration in Electron Optics," *Adv. in Optical and Electron Microscopy,* Academic Press, 1966.

[E4] Ohiwa, H., E. Goto and A. Ono, "Elimination of Third-order Aberrations in Electron-Beam Scanning Systems," *Trans. IECE Japan,* 54-B, 1971, p. 730, (in Japanese). Also in *Electronics and Communications in Japan,* 54-B, 12, 1971, p. 44.

[E5] Owen, G. and W. C. Nixon, "Aberration Correction for Increased Lines Per Field in Scanning Electron Beam Technology," *J. Vac. Sci. Technol.* 10, 1973, p. 983.

[E6] Munro, E., "Calculation of the Optical Properties of Combined Magnetic Lenses and Deflection Systems with Superimposed Fields," *Optik,* 39, 1974, p. 450.

[E7] Munro, E., "Design and Optimization of Magnetic Lenses and Deflection Systems for Electron Optics," *J. Vac. Sci. Technol.* 12, 1975, p. 1146.

[E8] Pfeiffer, H. C., "New Imaging and Deflection Concept for Probe-forming Microfabrication Systems," *J. Vac. Sci. Technol.,* 12, 1975, p. 1170.

[E9] Goto, E. and T. Soma, "MOL (Moving Objective Lens) Formulation of Deflective Aberration Free System," *Optik* 48, 1977, p. 255.

[E10] Hawkes, P. W., "Computer Calculation of Formulae for Electron Lens Aberration Coefficients," *Optik* 48, 1977, p. 29.

[E11] Soma, T., "Relativistic Aberration Formula for Combined Electric-Magnetic Focusing-Deflection System," *Optik,* 49, 1977, p. 255.

[E12] Hearn, A. C., "Reduce 2 User's Manual," Univ. Utah, Salt Lake City, Utah, 1974.

Beam broadening

[B1] Boersch, H., "Experimentelle Bestimmung der Energieverteilung in thermisch ausgelösten Elektronenstrahlen," *Z. Phys.* 139, 1954, p. 115.

[B2] Zimmermann, B., "Broadened Energy Distribution in Electron Beams," *Adv. Electronics and Electron Phys.* 29, 1970, Academic Press, p. 251.

[B3] Pfeiffer, H. C., "Experimental Investigation of Energy Broadening in Electron Optical Instruments," *11th EIL Beam Symp.,* 1971.

[B4] Pfeiffer, H. C., "Basic Limitation of Probe Forming Systems due to Electron-Electron Interaction," *Proc. 5th SEM Symp.,* 1972.

[B5] Ichinokawa, T., "Effect of Electron Source to Energy Resolution in Electron Velocity Analysis—Interpretation of Boersch Effect," *JJAP* 8, Feb. 1969, p. 137.

[B6] Ditchfield, R. W. and M. J. Whelan, "Energy Broadening of the Electron Beam in the Electron Microscope," *Optik* 48, June 1977, p. 163.

Fly's eye lens optics

[F1] Newberry, S. P., T. H. Klotz, Jr. and E. C. Buschmann, "Advances in Fly's Eye Electron Optics," *Proc. Nat'l. Electronics Conf.* 23, 1967, p. 76.

[F2] Goto, E., T. Soma and A. Ono, "Ultra-high Precision Cathode Ray Tube," *J. Inst. TV Engrs. Japan* 26, 1972, p. 21, (in Japanese).

[F3] Shibata, S., T. Soma, E. Goto and A. Ono, "A New Type of Cathode Ray Tube Suitable for Bubble Chamber Film Measurements," *Nucl. Instr. Methods,* 123, 1975, p. 431.

[F4] Hughes, W. C., C. Q. Lemmond, H. G. Parks, G. W. Ellis, G. E. Possin and R. H. Wilson, "BEAMOS—A New Electronic Digital Memory," *AFIPS Conf. Proc.* 44, 1975, p. 541.

[F5] Speliotis, D. E., "Bridging the Memory Access Gap," *AFIPS Conf. Proc.* 44, 1975, p. 501.

[F6] Ono, A., M. Fukawa, H. Kichimi, H. Kodama, T. Murakami, R. Sugahara, A. Suzuki and K. Takahashi, "An Automatic Film Scanner (KAMA) Utilizing a New Type CRT with Double Deflection Scheme," *Nucl. Instr. Methods,* 141, March 1977, p. 193.

[F7] Parks, H. G., "Matrix Lens Electron Beam Recording Systems," *14th EIP Beam Symp.,* May 1977.

Semiconductor technology in Japan

by TAKUO SUGANO
The University of Tokyo
Tokyo, Japan

INTRODUCTION

The present status of semiconductor technology in Japan will be reviewed with special emphasis on research and development of semiconductor devices and of large scale integrated circuits for computers. First of all, major programs of R&D of semiconductor devices and large scale integrated circuits will be briefly described and then important achievement, especially on semiconductor devices for high speed digital application and on large scale integrated circuits for logic and memory use, will be discussed.

MAJOR RESEARCH PROGRAMS

Research and development of semiconductor devices and large scale integrated circuits are active in industry, governmental and semi-governmental laboratories and universities.

The Electrical Communication Laboratories, Nippon Telegraph and Telephone Public Corporation (NT&T) and the Electrotechnical Laboratory, Agency of Industrial Science and Technology, are significantly contributing to R&D of LSI through their own work and also through their projects, although their activity is not limited to LSI technology. NT&T is stimulating R&D in industry through their technical innovation such as electronic telephone exchange, data communication and picture transmission. The Electrotechnical Laboratory, too, is promoting R&D in industry through governmental projects such as pattern recognition.

In March 1976, VLSI (Very Large Scale Integration) Technology Research Association was established by the seven corporations in Japanese computer industry, such as Computer Development Laboratories, Inc., Fujitsu Limited, Hitachi, Ltd., Mitsubishi Electric Corporation, NEC-Toshiba Information Systems Inc., Nippon Electric Co. Ltd., and Tokyo Shibaura Electric Co. Ltd.

This is based on the four year national project for developing VLSI technology in cooperation with the Electrotechnical Laboratory and NTT. R&D is carried out at three laboratories: Cooperative Laboratories, responsible for fundamental technologies and Computer Development Laboratories, Inc. and Laboratories of NEC-Toshiba Information Systems Inc. both responsible for application technologies.

The total budget for R&D is approximately 70,000 million Yen (Approx. $290 million) and R&D subjects are covering (1) microfabrication technology, (2) crystal technology, (3) design technology, (4) process technology, (5) test and evaluation technology, and (6) device technology.

The Science and Technology Agency, which is sponsoring the big national projects such as space science and technology and atomic energy, is exploring the possibility of developing various subjects. The Agency picked up the subject "III-V Semiconductor for Functional Devices" and financed the Electrotechnical Laboratory and others for investigation and research on Gunn effect digital devices for computers.

The Ministry of Education, Research and Culture is granting financial aid for special research programs besides financing university for research done by individual staff or small group of staffs.

One of the programs is "Surface Electronics," new surface and thin film technologies of compound semiconductors, and includes R&D of technology for insulator, mainly oxide film formation on the surface of GaAs wafers and of its application to surface passivation of GaAs devices and to fabrication of GaAs MOSFETs and integrated circuits. This was a three year project and terminated in March 1978.

SILICON DEVICES

Many technological advances have been made to improve high frequency characteristics of bipolar transistors and MOSFETs. Smart self-alignment technique was successfully applied to MOSFETs as DSA (Diffusion-Self-Aligned) and to bipolar transistors as SET (Stepped Electrode Transistor) to avoid critical mask alignment. Various types of DSA MOSFETs have been announced as a high speed, sub-micron channel device with high drain avalanche breakdown voltage and without punch-through effect. Cut-off frequency of about seven GHz was reported even in the first experimental device by the Electrotechnical Laboratory.[1]

In order to raise the maximum oscillation frequency of bipolar transistors, very small inter-electrode gap between the base electrode and the emitter was realized by a self-alignment technique as shown in Figure 1. The Electrical Communication Laboratories called this transistor as SET and reported that the gap less than 0.4 μm was achieved and the cut off frequency of $|S_{21}{}^{e}|^2$ has reached to 8.4 GHz, although precise mask alignment whose tolerance was smaller than 0.3 μm was not used.[2]

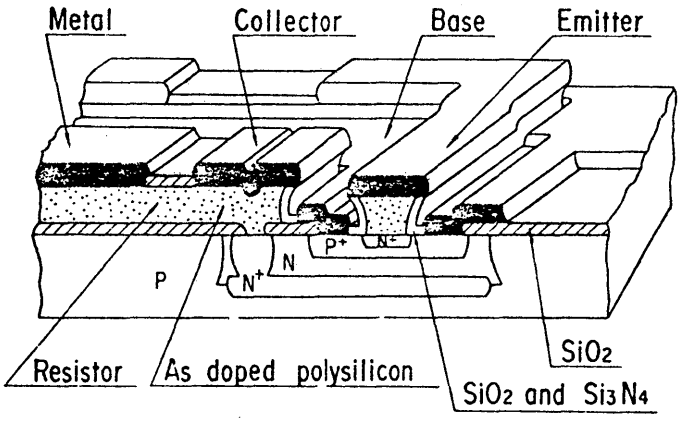


Figure 1—Structure of E^2IC

SIT (Static Induction Transistor) was proposed and developed in Tohoku University. This has a very short channel with high resistivity as illustrated in Figure 2 and triode-like current-voltage characteristics. The delay-time-power product is small and less than 0.01 pJ.[3] This device is useful also as a high output power transistor at microwave frequencies and Mitsubishi announced 20wSIT at 1GHz.[4]

GaAs DEVICES

Gunn effect functional devices with Schottky control gates, developed in the University of Tokyo and other cooperating organizations, are expected to have very small delay time-power product[5] such as 0.001 pJ for $Ga_{0.82}In_{0.18}Sb$.

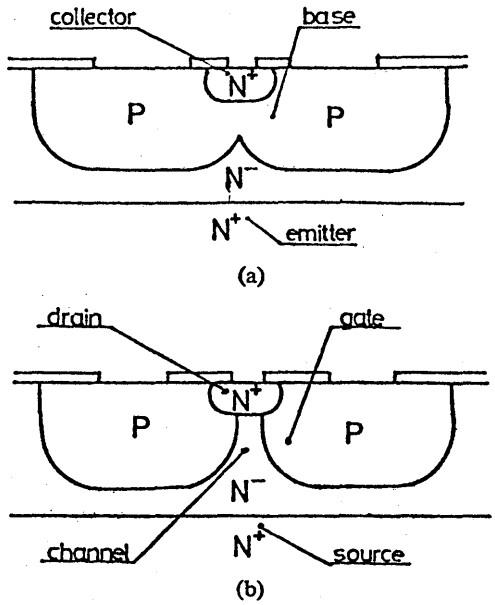


Figure 2—The structures of new transistor types made by lateral p-diffusion.
(a) bipolar transistor
(b) static induction transistor

This motivated the extensive study of material processing technology on III-V semiconductors in Japan.

The special research program "Surface Electronics" stimulated the research of technology of oxide film formation on the surface of GaAs wafers in universities. A group at Hokkaido University worked on anodic oxidation in the mixture of tartaric acid, glycol and water[6] and another group at the University of Tokyo worked on anodic oxidation in oxygen plasma.[7] In addition to R&D in universities, Matsushita developed the technique of thermal oxidation of GaAs in reduced ambient pressure[8] and Fujitu reported GaAs MOSFETs with the maximum oscillation frequency of 13 GHz using plasma oxidation technique.[9]

HIGH SPEED LOGIC

Logic integrated circuits with higher speed with more density of integration have been achieved by new circuit configuration and devices with better performance capability, supported by novel process technology.

The Electrical Communication Laboratories developed the new technologies, called N±IPOS(N^+ Insulation by Oxidized Porous Silicon) and E^2IC (Elevated Electrode Integrated Circuit) process to minimize the isolation area and isolation capacitance and to make the size of bipolar transistors smaller. Both N±IPOS and E^2IC are self-aligning technologies and the former is associated with isolation layer formation and base diffusion and the latter is effective in electrode formation.

Non-threshold logic proposed from the Electrical Communication Laboratories is a unique circuit configuration where bipolar transistors are operating in the active region only and never go to the saturation region. Therefore NTL IC fabricated by E^2IC technology showed the very short delay time per gate of 85 psec and the very small delay time-power product of 0.075 pJ.[10] In 8 Bit ALU with 180 gates integrated in a chip of 1.6×1.0 mm^2 the delay time was 260 psec per gate.[11]

Using the configuration similar to I^2L, SIT logic IC showed the power consumption of 7 μw per gate and delay time of 30 nsec per gate and the smallest delay time-power product of 2 fJ.[12]

Improvement of I^2L using dielectric isolation or vertical pnp injector has been done by the Electrical Communication Laboratories and Mitsubishi. The former reported the delay time of six nsec and the delay time-power product of 0.3 pJ for three collectors configuration[13] and the latter announced the delay time power product of 0.07 pJ.[14]

A new logic circuit configuration, Wired OR Wired AND Logic (W^2L) was announced from OKI. Figure 3(a) illustrates an inverter of W^2L. Both A and B are the outputs due to an input, but the directions of the output current flow are opposite to each other as shown by the arrows. The output current of A terminal exists only when the logical output status is 'H' and of B only when the logical output status is "L." Figure 3(b) shows a circuit configuration corresponding to Figure 3(a). The performance of W^2L as shown in Table I is just intermediate between TTL and I^2L.[15]

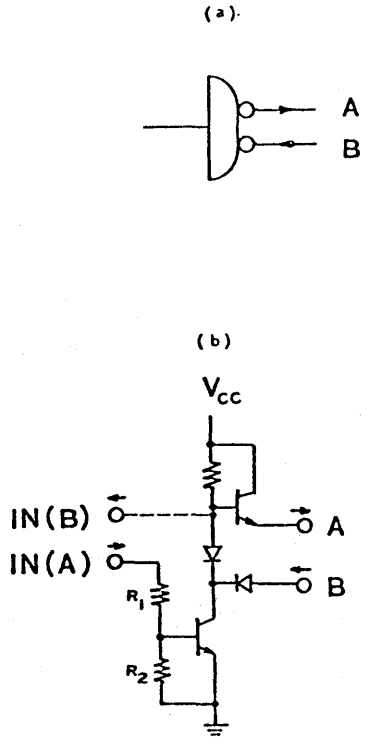


Figure 3—Basic inverter of W^2L

Four bits ALU composed by n channel enhancement type DSA MOSFETs with n channel depletion type MOSFETs as load showed 2.9 nsec as the delay time per gate and 0.71 mW as the power consumption per gate which was reported by NEC and the Electrotechnical Laboratory.[16]

Recently Mitsubishi announced DSA masterslice LSI for 920 gates random logic, whose performance is listed in Table II.[17]

Punch through mode operation of a submicron gate MOSFET is useful because of its inherent high speed performance and low output impedance. Fujitsu proposed to use this mode of operation for high speed logic. An experimental 13 stage ring oscillator consisting of E/D inverters and buffer circuit by 1 μm gate MOSFET (effective gate length of 0.5 μm) with 350 Å gate oxide film showed 90 ps switching delay and an E/E inverter indicated the delay time-power product of 2 fJ.[18]

TABLE I

	T^2L	W^2L	I^2L
Delay Time ns/gate	3-30	3-30	10-100
Power Consumption mW/gate	2-20	1-10	0.01-1
Threshold Voltage V	1.4	0.7-1.4	0.7
Power Supply V	5	1.6-5	0.8-5
Load Drive	good	good	poor
Mask	6	6	4
Transistor Mode	normal	normal	inverse
Active Area Ratio /gate	1	0.4	0.1
/ALU	1	0.3	0.2

TABLE II—Feature of the DSA MOS Masterslice Chip

Number of Circuits	800 internal gate cells 116 output gate cells	
Power Supplies	V_{cc} 5V single power supply	
Propagation Delay Time	basic cell (minimum interconnection)	1.0 ns at 3.6 mW
	basic cell (average interconnection of ALU)	3.0 ns at 3.6 mW
Output buffer	TTL compatible t_r 7 ns at 1 TTL load t_f 5 ns at 1 TTL load	
Total power dissipation	internal gate cells	2.5 W
	output buffer circuits	0.5 W
Chip size	7.68×7.88 mm²	
Number of pins	maximum 120 pins	

Toshiba paid much effort for developing SOS devices and reported 1300 gates LSI by n channel Si gate, E/D configuration. This LSI RFO (Register File O) is composed of 250 bits static RAM and peripheral circuits. This chip size is 4.14×4.47 mm² and the delay time per gate is 0.70 nsec and the access time is 65 nsec, the power consumption is 450 mW and the delay time-power product per gate is 0.21 pJ.[19] R&D of GaAs ICs were reported, too.

Normally Off-Type GaAs MESFETs were used for high speed logic by Fujitsu. An experimental 13 stage ring oscillator showed the delay time of 280 psec and the delay time-power product of 74 fJ.[20]

Significant progress of Gunn effect functional devices has been made by the Electrotechnical Laboratory and Fujitsu. Fujitsu announced two bits shift register which could be operated at the clock rate of 1.6 GHz.[21] Recently 20 ps/gate Gunn-effect high speed carry finding device for 8 bit binary adder was released by the Electrotechnical Laboratory.[22]

MEMORY IC

Significant reduction of power consumption in bipolar RAM was achieved. Fully static 4 K bipolar TTL RAM whose typical access time is 25 nsec and power consumption is as low as 350 mW has been developed by Hitachi. To reduce power consumption, stand-by current of a cell is decreased to 4 μA and it is increased by sinking current through lower word line at the instant when the cell is going to be selected. The fabrication process includes oxide isolation with double layer metallization and transistors with the emitter of 3 μm×3 μm in size have low parasitic capacitances (C_{TS}=0.13 pF, C_{TC}=0.041 pF and C_{TE}=0.034 pF).

The chip size is 3.4 mm×6.3 mm.[23] Similar 4 K static bipolar TTL RAM was announced by NEC. The access time is 40 nsec and the power consumption is 500 mW. The size of a memory cell is 38 μm×60 μm and of a chip is 3.24 mm×5.3 mm. Novel polycrystalline silicon self-aligned

(PSA) technique in combination with non-epi technology (diffused collector) and LOCOS process was used.[24]

The Electrical Communication Laboratories released a 64 K bit MOS RAM whose access time is 200 nsec and power dissipation is 150 mW. MOSFETs whose effective channel length is 2 μm, gate oxide thickness is 500 Å, junction depth is 0.25 μm and threshold voltage is 0.8 V at 1 V drain voltage are used. The technologies: As doped polysilicon gate, selective oxidation, Mo metallization, all ion-implantation, flowed phospho-silicate glass and fine pattern technology, 2 μm pattern width, are incorporated. The cell size is 14 μm×15 μm and the chip size is 6.1 mm×5.8 mm.[25]

Clocked CMOS 4 K static RAM, measuring 4.7 mm square, has been developed by Toshiba. Conventional Si gate technology and fine patterning technology of 5 μm were used and by narrowing the width of the boundary region, the cell area of 47.0 μm×68.0 μm was achieved. The access time is 200 nsec, and the operational power dissipation is 50 mW at the cycle time of 1 μsec and the stand-by power dissipation is 0.5 μW.[26] Recently Toshiba reported new multiplexed electrode per bit structure for a 64-K bit charge-coupled-device memory, where a merging serial register is combined with an ME/B array to make a practical and flexible CCD array. The memory operates typically at 5 M bits/sec data rate, while a 512-bit test array is operated in less than 140 nsec transfer execution time.[27]

Non-volatile semiconductory memory has been developed, too, as an erasable read only memory (EROM), an electrically alterable read only memory (EAROM), and a non-volatile random access memory (NVRAM). Two types of non-volatile semiconductor memory, which are a metal-insulator-oxide-silicon (MIOS) structure and a floating gate structure, were pursued.[28] As MIOS devices MAOS,[29] MTa_2O_5-Al_2O_3 OS[30] and MOSOS[31] were studied. As floating gate devices FAMOS is the first one, but it has disadvantages such that the controllability of stored charge amount in writing operation is not sufficient and the electron injection ratio into the floating gate is rather small. These result in longer programming time. However these disadvantages have been removed by introducing avalanche injection with stacked gate structure (SAMOS),[32] which was developed by Toshiba.

Recently Toshiba announced 2 K bit electrically alterable avalanche-injection type ROM with stacked-gate structure. The writing time is 20 μsec for a single transistor and is less than 5 sec for a fully decoded 2048-bit memory. Erasure of the memory is accomplished either by ultra-violet light irradiation onto the floating gate or by electric field emission of electrons from the floating gate to the control gate.[33]

The combination of hot electron injection and direct tunneling injection in p-channel MNOS memory transistor was used to make a 1024 bit RAM, whose chip size is 3.60×3.61 mm^2, and read access time is 600 nsec and write cycle time is 10 μsec-100 μsec by Toshiba.[34]

Recently a single 5 V supply 16 K EPROM (Erasable Programmable Read Only Memory) using DSA technique and self-aligned gate technology was announced by NEC. The access time is 300 nsec, the power consumption is 450 mW and the chip size is 3.6 mm×4.98 mm.[35]

CONCLUDING REMARKS

It should be notified that:

(1) self-alignment technique is successfully used to fabricate devices such as DSA and SET, and
(2) much efforts are paid to develop new devices like SIT and Gunn effect digital devices.

Of course steady improvement of conventional integrated circuits including devices used in them is going on and this is backed up with new process, technology and new circuit configuration.

ACKNOWLEDGMENT

The author would like to thank Cooperative Laboratories of VLSI Technology Research Association, the Electrical Communication Laboratories, Fujitsu, Hitachi, Mitsubishi, Nippon Electric, OKI, and Tokyo Shibaura Electric for permitting him to include their published and/or unpublished data in this manuscript. Otherwise this review would not have been possible.

REFERENCES

1. Tarui, Y., Y. Hayashi and T. Sekigawa, *Proc. 1st Conf. Solid State Devices*, Tokyo, 1969, p. 105.
2. Sakai, T., Y. Sunohara, Y. Sakakibara and J. Murota, *Proc. 8th Conf. Solid State Devices*, Tokyo, 1976, p. 43.
3. Nishizawa, J. and R. M. Wilamowski, *Proc. 8th Conf. Solid State Devices*, Tokyo, 1976, p. 151.
4. Kajiwara, Y., G. Nakamura, Y. Higaki and M. Ooga, National Convention of Semiconductor Division, The Inst. of Electronics & Elect. Comm. Engrs. Japan, 1977, S4-6.
5. Sugeta, T., M. Tanimoto, T. Ikoma and H. Yanai, *IEEE Trans. Electron Devices*, ED-21, 1974, p. 504.
6. Hasegawa, H. and H. L. Hartnagel, *Jour. Electrochem. Soc.* 123, 1976, p. 713.
7. Yamasaki, K. and T. Sugano, *Proc. 9th Conf. Solid State Devices*, Tokyo, 1977.
8. Private communication.
9. Mimura, T., N. Yokoyama, Y. Nakayama and M. Fukuta, *Proc. 9th Conf. Solid State Devices*, Tokyo, 1977.
10. K. Kato, National Convention of Semiconductor Division, The Inst. of Electronics & Elect. Comm. Engrs. Japan, 1977, p. 63.
11. Sakai, T., Y. Sunohara, H. Nakamura and T. Sudo, *ISSCC Digest of Technical Papers*, 1977, p. 196.
12. Nishizawa, J., M. Shinbo and T. Oomi, National Convention of Semiconductor Division, The Inst. of Electronics & Elect. Comm. Engrs. Japan, 1977, p. 291.
13. Yokoyama, K., A. Yoshii, T. Adachi and T. Sudo, National Convention of Semiconductor Division, The Inst. of Electronics & Elect. Comm. Engrs. Japan, 1977, p. 73.
14. Nakano, T., Y. Horiba, K. Kijima, K. Tanaka and O. Tomisawa, *Proc. 8th Conf. Solid State Devices*, Tokyo, 1976, p. 129.
15. Ishino, H., National Convention of Semiconductor Devision, The Inst. of Electronics & Elect. Comm. Engrs. Japan, 1977, p. 377.
16. Ohta, K., M. Morimoto, M. Saitoh, T. Fukada, A. Morino, K. Shimizu, Y. Hayashi and Y. Tarui, *ISSCC Digest of Technical Papers*, 1975, p. 124.
17. Nakano, T., O. Tomisawa, K. Anami, M. Ohmori, I. Ohkura and M. Nakaya, *ISSCC Digest of Technical Papers*, 1978.

18. Nakamura, T., M. Yamanoto, H. Ishikawa and M. Shinoda, *ISSCC Digest of Technical Papers,* 1978.
19. Tango, H., K. Naeguchi, E. Sugino, S. Taguchi and T. Sato, *Technical Digest SSD77-32,* Inst. of Electronics & Elect. Comm. Engrs., Japan.
20. Ishikawa, H., H. Kusakawa, K. Suyama and M. Fukuta, *ISSCC Digest of Technical Papers,* 1977, p. 200.
21. Isobe, T., S. Yanagisawa and T. Nakamura, *Proc. 8th Conf. Solid State Devices,* Tokyo, 1976, p. 135.
22. Hashizume, N., S. Kataoka and K. Tomizawa, to be published in *Electronics Letters.*
23. Hotta, A., Y. Kato, K. Yamaguchi, N. Honma and M. Inadachi, *ISSCC Digest of Technical Papers,* 1978.
24. Okada, K., K. Aomura, J. Nokubo and H. Shiba, *ISSCC Digest of Technical Papers,* 1978.
25. Arai, E. and N. Ieda, *3rd ESSCIRC,* 1977, p. 74.
25a. Yoshimura, H., M. Hirai, T. Asaoka and Y. Toyoda, *ISSCC Digest of Technical Papers,* 1978.
26. Ochii, K., Y. Suzuki, M. Ueno, K. Sato and K. Asahi, *ISSCC Digest of Technical Papers,* 1977, p. 18.
27. Kohyama, S., H. Hatano, T. Tanaka and N. Kubota, *1976 IEDM Technical Digest,* p. 11.
28. Nishi, Y. and H. Iizuka, *Proc. 8th Conf. Solid State Devices,* Tokyo, 1977, p. 191.
29. Nakanuma, S., T. Tsujide, R. Igarashi, K. Onoda, T. Wada and M. Nakagire, *ISSCC Digest of Technical Papers,* 1970, p. 68.
30. Matsuo, T., *Japan Jour. Appl. Phys.* 12, 1973, p. 1862.
31. Horiuchi, M., *IEDM Abstract,* 1972, p. 242.
32. Iizuka, H., F. Masuoka, T. Sato and M. Ishikawa, *IEEE Trans.* ED, ED-23, 1976, p. 399.
33. Iizuka, H., F. Masuoka, T. Sato and M. Ishikawa, *IEEE Trans.* ED, ED-23, 1976, p. 379.
34. Uchida, Y., N. Endo, S. Saito, M. Konaka, I. Nojima, Y. Nishi and K. Tamaru, *IEEE Jour. Solid State Circuits,* SC-10, 1975, p. 288.
35. Kikuchi, M. et al., private communication.

The development of computers in Japan

by OSAMU ISHII
Electrotechnical Laboratory
Tokyo, Japan

INTRODUCTION

Japan's research and development of computers began in the 1940's. However, Japanese production of commercial computers started a decade after the U.S. production did. Since then the computer industry has developed into the most rapidly growing industry in Japan. Thirty-nine thousand computers were installed as of the end of 1976. This is second in the world after the United States.

It is difficult to cover all the computer industry and its R&D activities. The aim of this paper is to give a brief review of Japanese computer technology, with a short historical background, and a description of present status and some ongoing developments.

DEVELOPMENT OF JAPANESE DIGITAL COMPUTERS

The research on computers in Japan began at universities and national laboratories. Japan's first digital computer, ETL Mark-1, a small scale experimental relay computer, was completed at the Electrotechnical Laboratory of the Japanese Government, in 1952.[1] Then the relay computers, FACOM 100[2] and ETL Mark-2[1] were constructed in 1954 and 1955 respectively. These machines were used for Japan's first computing services.

The first stored program computer in Japan, called Fujic, which employed vacuum tubes and a mercury delay-line storage, was completed in 1956.[3] The research and development of transistor computers started almost at the same time and ETL Mark-3, a small experimental computer was also completed in 1956.[4] ETL Mark-4,[5] which followed Mark-3, was constructed in the next year, and the technologies developed were transferred to many types of commercial computers.

In the same period, the parametron,[6] a logical element using the phenomenon called the parametric oscillation, was invented, and parametron computers PC-1 (1958),[7] and M-1 (1957)[8] were constructed at the University of Tokyo and the Electrical Communication Laboratory (ECL) of Nippon Telegraph and Telephone Public Corporation (NTT). Several types of commercial computers later employed the technologies developed for these pilot computers.

Japanese companies began to manufacture and market computers around this time. In the 1960's computers began to be introduced into various fields, but at the time Japan was said to be ten years behind the U.S. in technology and industrial management. Faced with this situation, the Japanese Government and computer industry realized that an extraordinary effort was required to close the gap. At that time the "National Research and Development Program" was established in 1966, and the "Super High-Performance Computer Development Project"[9] was carried out by the Agency of Industrial Science and Technology of the Ministry of International Trade and Industry (MITI). The project was completed in 1971, and the technologies developed by the project were applied to several new commercial computer series in Japan.

In the later half of the 1960's, the era of third generation computers, domestic computer manufacturers announced their commercial series of computers. They were: FACOM 230 Series (Fujitsu), HITAC 8000 Series (Hitachi), NEAC 2200 Series (Nippon Electric, NEC), TOSBAC Series (Toshiba), MELCOM Series (Mitsubishi) and OKITAC Series (Oki).

In 1968, NTT started the development of DIPS-1 (Dendenkosha Information Processing System-1),[10] which is a large scale computer designed for a nation-wide data communication service. DIPS-1 started offering the computing service in December 1973.

Following DIPS-1, NTT developed in co-operation with the domestic mainframe manufacturers the DIPS-11 series which included three models.

In 1972, under the promotion of the Japanese Government, the six domestic computer manufacturers were reorganized into three groups, and each group started the development of a new computer series. These series were called "M Series" (Fujitsu-Hitachi group), "ACOS Series" (NEC-Toshiba group) and "COSMO Series" (Mitsubishi-Oki group). The completion of all models of these series from large scale computers to small computers had been announced by the spring of 1977.

COMPUTER INDUSTRY IN JAPAN

The commercial production of electronic computers in Japan was started in the late 1950's by electronics and telecommunications equipment manufacturers, as opposed

to business equipment manufacturers in the United States. Another point of difference between U.S. companies and Japanese companies is that the latter started with the production of solid-state computers skipping over vacuum tube systems. At that time, however, the computer industry was very new to the Japanese, not only from the point of view of technology but also from that of rental systems. The Japanese Government recognized the importance of this field of industry, and the Japan Electronic Computer Company (JECC) was established with the support of MITI by domestic computer manufacturers in 1962 for the purpose of establishing a rental system in Japan. Through the 1960's, the Japanese computer industry has progressed remarkably, especially in the later half of the decade, and application of computers has penetrated many sectors of Japanese society. Figure 1 shows the trend of computer production after 1965 and the trends of peripherals, terminals and small business computers in Japan. The growth of the production of terminals reflects the rapid growth of on-line systems in various fields of computer applications, such as banking, seat reservations and inventory control systems.

Table I shows the composite ratio of applications of on-line systems in Japan.[11]

Besides domestic computer manufacturers, there are many foreign manufacturers in the Japanese computer market. For instance, IBM and NCR have established their fully owned subsidiaries, and other companies are carrying out technical and sales activities in joint-ventures with Japanese firms. Others have sales offices in Japan.

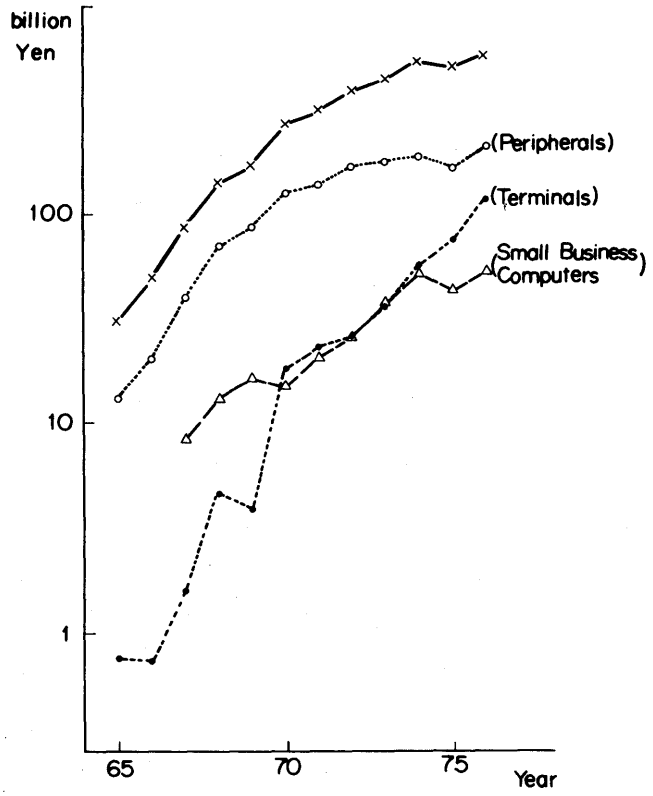


Figure 1—Production of computer systems in Japan (Source: MITI)

TABLE I.—Applications of On-Line Systems

Fields	Composition (%)
Finance, Securities, Insurance	21.3
Manufacturing, Sales, Stocks	43.8
Computing Service	8.1
Pollution Monitoring	7.3
Transportation, Traffic Control	5.3
Others	14.2
Total	100.0

The share of domestic and foreign products in the Japanese market are shown in Table II, as of the end of December 1976. In terms of total system value, domestic production accounts for 56.8 percent, and imports for 43.2 percent. But for large scale systems, imports exceed domestic production.

The market for small and very small computers is growing and they are penetrating the large computer market rapidly because of their cost performance. "Office computers" or "Office systems" (small business computers) are becoming feasible for small enterprises, and sales in 1977 are expected to reach 100 billion yen.

Microcomputer applications are expected to increase markedly, not only for computers, but also for various types of peripheral controls and equipment controllers. Sales are expected to be 20 billion yen for the 1977 fiscal year.

Japan has prudently liberalized capital transaction and trade regulations for the data processing industry. Although surrounded by controversy regulations for hardware were liberalized up to 100 percent in December 1975 and for software in April 1976.

DEVELOPMENT OF 3.5 GENERATION COMPUTERS

The three groups of the six domestic computer manufacturers started to develop three new 3.5 generation computer series under MITI's support in 1972. Since 1974, each group announced new computer models of their series one after another and all three completed their series by the spring of 1977.

The series are shown in Table III, with rough correspondence of the system scale between IBM System/370. Originally the cost/performance of these series surpassed that of the IBM System/370. In 1977, however, IBM announced the 303X processors, and cut the prices of the System/370, also the value of the yen increased, so the situation has changed.

Technological trends adopted in the new series computers are:

(1) Development of LSI and packaging technology
(2) Stress on IBM compatibility

TABLE II.—General-Purpose Computer Installations in Japan as of the end of December 1976 (Source: MITI)

System Scale*	Domestic Systems		Imported Systems		Total	
	No. of Sets	Value**	No. of Sets	Value**	No. of Sets	Value**
Large	1,247	703,634	932	761,885	2,179	1,465,519
Medium	4,617	456,967	1,465	183,938	6,082	640,904
Small	8,235	156,803	2,788	57,782	11,023	214,585
Very Small	10,985	68,461	8,658	52,485	19,643	120,946
Total	25,084	1,385,864	13,843	1,056,090	38,927	2,441,953

* System scale is classified by purchase price as follows:

Large : More than 250 million yen
Medium : 40 - 250 million yen
Small : 10 - 40 million yen
Very small : Less than 10 million yen

** Value; million yen, the conversion rates between Japanese Yen and U.S. Dollar are approximately 290:1 for 1976

(3) Virtual storage and multi-processing
(4) Considerations for data communication and data base oriented systems

Table IV summarizes the characteristics of several representative new series computers (large scale models).

Domestic mainframe manufacturers are also producing many kinds of peripherals including magnetic disk units similar to IBM's.

Many operating systems were developed. The operating systems for each series and model group are shown in Table V. The development of OS aimed at the realization of system functions and performance, which would equal or better IBM's MVS SVS, OS/VS1 or DOS/VS. The most important consideration in developing OS was compatibility with the operating systems which were being supported by computer manufacturers.

NTT's DIPS-1 was developed during the era of third generation computers. DIPS-11, which is a 3.5 generation version of the DIPS-1, consists of Model 10, Model 20 and Model 30. Model 30, the largest system of the series, surpasses the performance of the IBM System/370 Model 168. Table VI shows a summary of DIPS characteristics. These models have full compatibility with each other on software and peripheral equipment.

Another trend in data processing is distributed processing, or computer networks. In 1974, IBM announced SNA (System Network Architecture), and some extensions were introduced in 1976. After the introduction of SNA, most mainframe manufacturers announced their versions of network architecture. Japanese manufacturers for example announced, ANSA (Advanced Network System Architecture, Toshiba), DINA (Distributed Information Processing Network Architecture, NEC), MNA (Multishare Network Architecture, Mitsubishi), MSNA (M Series Network Architecture, Fujitsu-Hitachi), FNA (Fujitsu Network Architecture, Fujitsu), HNA (Hitachi Network Architecture, Hitachi) and DONA (Decentralized Open Network Architecture, Oki). NTT is now promoting DDX (Digital Data Exchange) and has also announced its own network architecture, DCNA (Data Communication Network Architecture). These network architectures are not yet well es-

TABLE III.—Correspondence of the New Computer Series with IBM Systems

IBM System/370	M Series	ACOS Series	COSMO Series
(Double to Triple of 168)	190	900 Model 1, 2	
168	180 180 II	800 Model 1, 2	
158	170 160	700 600	900
145, 148	160 II 160 S	500	700 II 700
135, 138	150	400	500
125	140	300	
115	130	200	300

TABLE IV.—Summary of Representative New Series Computers (Large Scale)

		M-190	M-180	ACOS-900	COSMO-900
CPU	Processors	1, 2	1, 2	1, 2, 3, 4	2
	Performance	155ns (ave.)	310 ns (ave.)	3.4 MIPS	0.6 MIPS
	Instructions	193	195	301	186
	Cache	16 KB	16 KB	16 KB	None
Main Memory	Capacity	1 - 16 MB	1 - 8MB	1 - 16 MB	0.5 - 4 MB
	Cycle time	480 ns	400 ns	600 ns	550 ns
	Width	32 B	8 B	8 B	8 B
	Interleave	2/4 way	4 way	2, 4, 8 way	2 way
Channel	Channels/CPU	16	16	18	6
	Throughput	20 MB/S	16 MB/S	60 MB/S	3.5 MB/S

tablished. However, they will play an important role in future computer systems in Japan.

RECENT TECHNOLOGIES AND PROJECTS

System architecture

An important goal of the development of the new computer series is realization of high OS efficiency for transaction processing and high level language oriented environments. Existing computer systems need a lot of supervisor processing for multi-processing control, which causes an increase in system overhead. The new process oriented architecture and firmware implementation for multi-programming control reduces system overhead, and increases system throughputs. Also, by adoption of the high-level language oriented architecture, instruction steps for execution of high level language statements were substantially reduced. Table VII shows the relative improvement of OS efficiency for the ACOS series.[12]

TABLE V.—Operating Systems Listed by Series and Group

M Series			ACOS Series		COSMO Series	
OSIV/F4	190, 180 II, 160	VOS 3	180, 170, 160 II	ACOS-6	900, 800, 700, 600	UTS/VS 900, 700 II, 700
OSIV/X8	190, 180 II, 160, 140	VOS 2	180, 170, 160 II, 150			
OSIV/F2	180 II, 160, 140, 130	VOS 1	160 II, 150	ACOS-4	500, 400, 300	RBM 700 II, 700
		EDOS-MSO/M	180, 170, 160	ACOS-2	200	UPS 500, DPS 300

TABLE VI.—Summary of DIPS-1 and 11

		DIPS-11 (1975-76)	DIPS-1 (1971)
Relative Performance		Model 10 20 30 1 1.5 3	1
CPU	Processors	Max. 2	Max. 4
	Instructions	169	160
	Cache	8/16 KB	8/16 KB
	Logic device	MSI/LSI	SSI/MSI
Main Memory	Capacity max.	8 MB, 16 MB	16 MB
	Increment	1 MB	1 MB
	Device	MOS·LSI	Core
Channel	Channel controllers	Max. 2, Max. 4	Max. 6
	Channels/controller	Max. 16	Max. 16
	Throughput/controller	Max. 12 MB/S	Max. 12 MB/S

TABLE VII.—Improvement of OS Efficiency

	System A	System B
I/O Overhead (Instruction step)	1	0.65
Instruction per Function (COBOL)	1	0.54

System A: Existing Architecture

System B: Process and high-level language oriented architecture

Virtual storage

The concept of virtual storage became common after the IBM System/370 adopted it. The M, ACOS and COSMO series employ the similar virtual storage technology. The details of the control such as page or segment sizes are different in each series. In some of the new series machines, channel address translation mechanisms are implemented by hardware. This was made possible by the progress in semiconductor technology.

Optional processors

The array processors such as the CDC STAR 100 or the CRAY-I are provided in the new series for a high-speed arithmetic processing option. These processors are designed for ease of use, for example, Hitachi's "IAP" allows standard FORTRAN programming.

LSI and package assembling

Semiconductor LSI's and LSI assembling techniques are the most important parts of modern computer hardware technology.

For instance, the floor space, weight and power consumption of the main frame of DIPS-11 (1975-76)[13] were reduced one fourth to one third, one third to two thirds, and a half to two thirds respectively from those of its predecessor, DIPS-1 (1971). These improvements were mainly realized by the progress in LSI technology and assembly techniques.

Typical LSI specifications used in the new series computers are shown in Table VIII. As an example, the logic LSI chip for the ACOS series,[14] is shown in Figure 2. In this case, the assembling has three distinct levels:

(1) Semiconductor LSI chip: LCML (LSI oriented low energy CML), max. 200 gates per chip, 7 pico joule per gate.

TABLE VIII.—Typical Specifications of LSI Used in M and ACOS Series

Item		M Series	ACOS Series
Logic	Chip size	4 x 4mm	3.2 x 3.2mm
	Gates/chip	100	Max. 200
	Delay time/gate	0.7ns	0.7ns
	Power dissipation/gate	35mW	10mW
Memory	Bits/chip	4,096	4,096
	Access time	100ns	Max. 200ns
	Cycle time	220ns	Max. 400ns
	Power dissipation/bit	0.12mW	0.11mW

(2) LSI package: 80mm × 80mm ceramic package, max. 110 LSI chips (max. 3,500 gates) per package, 240 connector pins. Forced air cooling.
(3) High density printed board: 530mm × 390mm, 15 conductor layers printed board, 12 LSI packages (max. 40,000 gates) per board.

Figure 3 is a photograph showing the LSI chips on the package.

Kanji input and output

The Japanese language is usually written in a mixture of "Kanji" (ideographic) and "Kana" (phonetic) characters. There are now only 48 phonetic characters, but several thousand ideographic characters are used in Japan (2,000-3,000 characters are popular, sometimes up to 10,000 characters are needed). Because the Japanese language is not

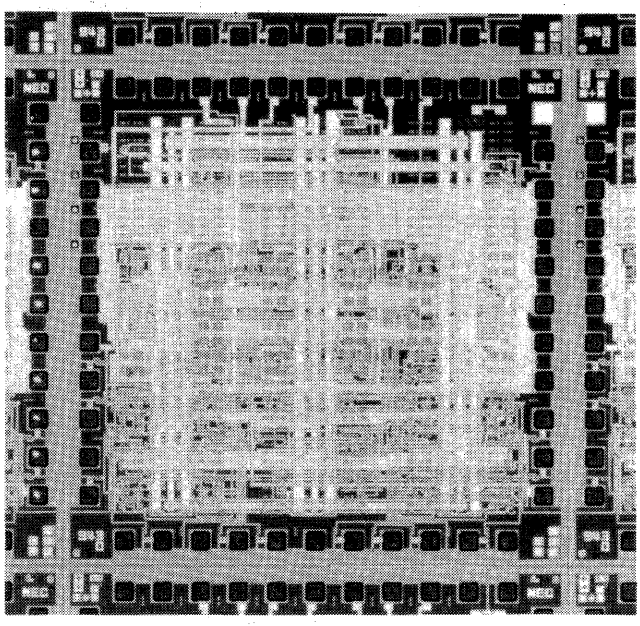


Figure 2—Logic LSI chip

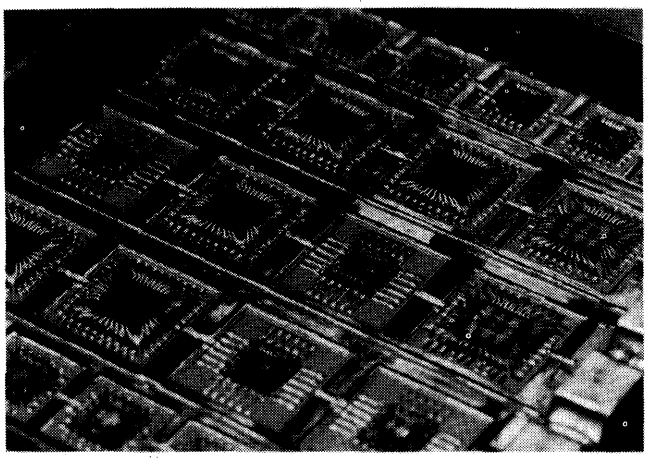


Figure 3—Logic LSI chips on the package

adaptable to a purely phonetic writing system "Kanji" are a most important medium for information exchange. However, computer systems today are not powerful enough to handle "Kanji." The main reason is lack of suitable input and output devices.

For output devices, expensive character generators are needed because of the huge character sets and complicated form of each character. Figure 4 shows an example of Kanji printer output[15] which includes "Kanji," "Kana" and some alpha-numeric characters represented by 24×24 dot matrix. Many types of memories have been used for Kanji font memories. However, the progress in LSI technology is reducing memory costs, so that one major problem, "Kanji" output, is expected to be solved in the near future.

"Kanji" input is a more difficult problem than output because of its man-machine interactions. A conventional "Kanji" typewriter has a great many character keys and several shift keys because of its large character sets. Accordingly, it is very difficult for even a skilled operator to use a "Kanji" typewriter.

Table IX is a comparison of typical characteristics of the English alphabet and "Kanji" I/O devices. The input speed of the "Kanji" device is about one fifth that of the alphabet in characters per minute. However, one "Kanji" character sometimes corresponds to one word so that the input speed of total information would be better than one fifth.

There are many types of "Kanji" selection mechanisms

２４×２４ドットマトリックス

３．７ｍｍ×３．７ｍｍ

漢字２０字／秒

Figure 4—An example of Kanji printer output

for the input device. Table X summarizes trade off between input speed and training necessary for various "Kanji" input devices and systems.

"Kanji" OCRs are being researched, in addition to OCRs for "Kana" and alpha-numeric characters now commercially in operation.

The VLSI project

Through the development of the 3.5 generation computers, it became clear that semiconductor technology, especially ultra-high density integration techniques, will play a key role in the future computer industry. In view of this situation, domestic computer manufacturers are emphasizing the "Very Large Scale Integration" (VLSI) project under MITI's support and guidance. The VLSI technology research association was started by a joint effort of five domestic mainframe manufacturers, NTT, and the government's ETL which provided technical guidance. The goal of this project is to develop the basic LSI and LSI production technologies for fourth generation computers. Production and process techniques for very precise (sub-micron) patterns using electron-beam, x-ray and conventional photolithography are being developed. From 1976 to 1979, a total of 70 billion yen is expected to be invested in this project.

The development program of pattern information processing systems

In today's "computerized society," computer utilization is becoming more and more varied, and increasingly higher efficiency and intelligence are expected. To cope with such requirements, it is desirable that future information systems advance into the area of pattern information processing.

The Japanese Government started a research and devel-

TABLE IX.—Comparison of Alphabet and Kanji I/O

	Character Set (C)	Dot Matrix (D)	Memory Capacity (C) x (D)	Code	Input Speed	Price ($) (Typewriter)
Alphabet (A)	26	5 x 7	910	1 Byte	250 ch./min. (50 words/min.)	100
Kanji (K)	4,000	24 x 24	2,304,000	2 Bytes	45 ch./min.	500
Ratio (K)/(A)	150	16	2,500	2	0.18 (0.9)	5

TABLE X.—Comparison of Kanji Input Systems

	Input Speed Characters/Min.	Training*
Kanji Keyboard with Multishift Keys	40 - 80	4
Office Kanji Typewriter	30 - 50	3
Kanji Tablet	30 - 60	2
Mnemonic Code Input System	60 -100	5
Interactive Input System	30 - 60	1
Kana-Kanji Conversion System		1

* 1: easy ⟶ 5: hard

opment program on "Pattern Information Processing Systems (PIPS)" in July 1971.

The research activities are:

(1) Development of new materials and devices which may be applicable to pattern information processing.
(2) Studies of subsystems for such visual and aural patterns as characters, speech, pictures and three dimensional objects, as well as for understanding natural language.
(3) Development of information systems having such new capabilities as parallel processing, associative information retrieval, and inference, or learning.

The activities in these research areas are closely related to each other. The final goal of the project is to construct a prototype system referred to as PIPS, which will be able to integrate the accomplishments attained in each area.

The pilot models for character recognition, picture and three dimensional object recognition, and word voice recognition, were demonstrated in July 1977.

Besides these pattern recognition subsystems, a high-performance LSI microprocessor,[16] multi-microprocessor systems, high-level language oriented machines, a magnetic bubble data base machine, and several other systems are under development.

CONCLUSION

Twenty years have passed since the commercial production of computers started in Japan. During this period the speed of technical development has been extremely rapid, and the applications of computers has increased greatly. It has affected almost every sector of society. The computer industry has become one of the most important industries in Japan. Now, there are new challenges facing the industry. In the next decade information processing systems which meet a wide variety of social needs must be developed. In these circumstances, technology which improves the cost/performance of computers will as always be sought after. But special applications related to traditional Japanese culture, mainly concerned with the language, will increase their role in Japanese society.

ACKNOWLEDGMENTS

The author appreciates the support in the preparation of this paper by the Japan Electronic Industry Development Association. Many organizations have supplied information, in particular Fujitsu, Hitachi, Nippon Electric, Mitsubishi Electric, Oki Electric, Toshiba and Nippon Telegraph and Telephone Public Corporation. The author is also appreciative of the advice of Professor H. Aiso of Keio University. Also, his thanks go to Mr. M. Meserve for his kind assistance in checking this manuscript.

REFERENCES

1. Goto, M., Y. Komamiya, R. Suekane, M. Takagi and S. Kuwabara, "Theory and Structure of the Automatic Relay Computer E.T.L. Mark II," Researches of the Electrotechnical Laboratory, No. 556, September 1956.
2. Matsuyama, T., "History of Fujitsu Computer Development," (in Japanese), *Journal of the Information Society of Japan,* Vol. 18, No. 7, 1977, pp. 664-674.
3. Okazaki, B., "Electronic Computer FUJIC and Examples of Calculations," (in Japanese), *Journal of the Institute of Electrical Engineers of Japan,* Vol. 40, No. 6, 1957, pp. 722-725.
4. Takahashi, S., H. Nishino and I. Matsuzaki, "ETL Mark-III, a Transistor Digital Automatic Computer," *Bulletin of the Electrotechnical Lab.,* Vol. 20, No. 4, 1956, pp. 279-291.
5. Nishino, H., S. Takahashi, I. Matsuzaki, H. Aiso, K. Kondo and H. Yoneda, "Transistor Computer ETL Mark IV," (in Japanese), *Journal of the Institute of Electrical Communication Engineers of Japan,* Vol. 42, No. 11, November 1959, pp. 1038-1045.
6. Goto, E., "The Parametron, a Digital Computing Element which Utilizes Parametric Oscillation," *Proc. IRE,* Vol. 47, 1959, p. 1304.
7. Takahashi, H., *Parametron Computers* (in Japanese), Iwanamishoten, Tokyo, 1968.
8. Muroga, S., and K. Takashima, "The Parametron Digital Computer MUSASHINO-1," *Trans. IRE,* EC-8, 1959, p. 308.
9. Nakazawa, K., K. Murata, K. Ishihara, H. Iwakami, H. Horikoshi, H. Nishino and K. Noda, "The Development of the High Speed National Project Computer System," *Proc. 1st USA-JAPAN Computer Conf.,* 1972, pp. 173-181.
10. Takashima, K., I. Toda, K. Arai and M. Yamada, "A Large-Scale Data Processing System: DIPS-1," *Proc. 1st USA-JAPAN Computer Conf.,* 1972, pp. 193-202.
11. Hirayama, H., "Review of Information Processing Techniques and its Applications in the Society," (in Japanese), *Journal of the Inst. of Electronics and Communication Engineers of Japan,* Vol. 60, No. 8, August 1977, pp. 860-867.
12. Mizuno, Y., and K. Shirai, "Performance Evaluation of a System Architecture," (in Japanese), *Nikkei Electronics,* No. 107, 1975, pp. 125-135.
13. Yoshida, S., A. Kawamata, M. Yamada and T. Kishigama, "Development of the DIPS-11 hardware system," (in Japanese), *Electrical Communication Laboratory Technical Journal,* Vol. 26, No. 3, 1977, pp. 817-833.
14. Ishii, Y., H. Kanai, N. Tanaka and T. Yoshida, "High Density Packaging Techniques for Large Scale Computers," (in Japanese), *Nikkei Electronics,* No. 158, 1977, pp. 100-119.
15. The printer was developed at the Electrical Communication Laboratory of NTT, 1977.
16. Iizuka, H., Y. Hayashi, K. Tamaru and H. Hara, "Development of a High-Performance Universal Computing Element," *AFIPS Conference Proceedings,* Vol. 47, June 1978.

Remote data processing in Japan

by KANJIRO KOSHI and KIMIO IBUKI
Nippon Telegraph and Telephone Public Corporation
Tokyo, Japan

INTRODUCTION

This paper introduces the development process, some recent features and the perspectives of remote data processing in Japan. The development process shows that remote data processing in Japan has maintained a steady progress in number of systems and in its application.

Recent features pertain to remote data processing in Japan, concerning recent application patterns. Also, a forecast is made of how the remote data processing in Japan will grow from the viewpoints of the boundary expansion as well as increases in technological capability.

REMOTE DATA PROCESSING CHANGES AND PRESENT SITUATION

A much more significant role will be played by remote data processing, in order both to cope with modern innovations and to meet requirements of sophisticated and complex business activities and management. Because remote data processing can process and transmit information. A long-term forecast has been announced wherein remote data processing will grow over 20 percent each year in the future.

Computer usage in our country began in 1955 when the Nomura Security Co., Ltd. and Tokyo Stock Exchange introduced UNIVAC-120. The first remote data processing system was MARS, used by Japan National Railways, and JALCOM, used by Japan Air Lines. Both were seat reservation systems. Their operation started in 1964. There were 40,000 general purpose computers installed as of March 1977; 3,000 computers are used for remote data processing, showing a steady growth year by year.

Remote data processing development background

The background of computer usage development and remote data processing in Japan is outlined from social and economic, technological and legislative standpoints.

Social and economic

As the result of technological innovation, Japan has attained decent growth during the 1960's and early 1970's.

Steel and petrochemical enterprises, electric power industries etc. rationalized both productivity improvement and managerial efficiency by introducing new plants and technology. The purpose was the extension of business scale and profit.

This economic growth may be partly due to the existence of abundant, cheaper and high quality labor force. The increase in demand for labor force, caused by the economic development, brought about not only full employment but also both a shortage of manpower and wage hikes. As a consequence, enterprises were obliged to make efforts toward saving manpower. Hence, remote data processing became recognized as a basic and indispensable tool, because it has sophisticated data transmission and processing functions.

Technological background

Generally speaking, computer manufacturing enterprises as a newcomer should have totalized and highly advanced technological ability. Almost all computer manufacturers were at the same time communication equipment manufacturers.

In the 1960's, NTT applied great efforts to provide nationwide direct-dialing telephone network through researching various technologies, such as electronic switching and multiplexing radio or carrier transmission by coaxial cables. Resulting from this research, NTT made various technological contributions and aids to communication equipment manufacturers. Furthermore, NTT vigorously promoted the DIPS development project during the latter half of the 1960's and early 1970's, with the cooperation of principal Japanese computer manufacturers. DIPS uses very large scale 3.5 generation computers for NTT's remote data processing use. Through this cooperation, NTT played a leading technological role and furnished significant impacts in pertinent technologies and their advance. The Japanese government has also supported Japan's computer industries since the early period, through various technological developments and research by government related laboratories on hardware and software. It was early recognized that computer industries will play a key role in changing the Japanese industrial structure into a knowledge intensive format.

TABLE I.—Computer Usage in Japan

(As of March)

Items		1968	1970	1972	1974	1976	1977
No. of Sets	(A)	3,558	6,718	12,809	23,443	35,305	40,719
General Purpose Computers		—	(37.9)	(35.1)	(35.9)	(17.3)	(15.3)
No. of Sets	(B)	—	217	476	1,029	1,871	3,052
Remote Data Processing		—	(56.1)	(46.9)	(42.0)	(24.7)	(63.1)
Ratio of On-line	(%)	—	3.2	3.7	4.4	5.3	7.5
Computers (B)/(A)							

Remarks: ( ) for percentage of growth rate compared with previous year.

Legislative background

Public telecommunication services providers vary according to countries. Japanese law dictates that these services should be provided only by NTT and KDD.

There may also be various ways to operate remote data processing services, combined computers and telecommunications. In Japan, they are regarded as one of major telecommunication services and considered to be operated in principle in a competitive market. Furthermore, it was considered desirable, for the progress of remote data processing technologies in Japan, that common carriers be permitted to operate those services. Thus, these common carriers were able to join that market under certain restrictions. In 1971, the Public Telecommunications Law, mainly for telephone and telegraph, was amended to legalize the provision of remote data processing. This amendment enabled not only common use of leased lines by a group of companies or the operation of remote data processing by private companies, but also their operation by NTT and KDD. It significantly contributed to the development of remote data processing in Japan.

Since then, remote data processing systems are provided in three ways. One is implementation by users. Second is implementation and vending by NTT or KDD. Last is implementation and vending by private companies. Each contributes to the development of remote data processing in cooperation, making the most of their individual characteristics.

Remote data processing changes and present situation

Computer usage in Japan

Table I shows the change in the number of general purpose computers and that of computers for remote data processing use. It can be seen that, during the past decade, each number has increased 11.4 times and 14 times, respectively. The general purpose computers growth rate has been 32 percent each year and that of computer for remote data processing use has been annually 46 percent during the same period. So, the remote data processing growth rate is larger than that of general purpose computers. This means steady progress in remote data processing, as is also shown in increasing ratio of on-line computers. This trend is very prominent, especially during these past few years.

Change in number of remote data processing systems

Table II shows the change in the number of remote data processing systems. As shown by the table, the number of systems now is 25 times larger than in 1969. The average growth rate is nearly 50 percent annually.

Remote data processing application areas

Remote data processing application areas are shown in Table III. The table shows that the proportion has hardly changed. About 65 percent of all remote data processing systems are occupied by finance, insurance, securities systems and production, inventory control or sales management systems. Therefore, remote data processing may be summarized as mainly used to process massive business administration created data in order to increase business efficiency or rationalize manpower. Furthermore, it will be clear that new type systems are growing because pollution control systems have increased about seven times and traffic control systems have increased four times in number.

SOME RECENT REMOTE DATA PROCESSING USAGE FEATURES

Upgrading initial systems

In Japan, banking organizations have been the major users of remote data processing since early years. Banking type systems still occupy a larger proportion of all the systems. The initial purpose of these systems was simply to improve business efficiency. However, they have been revised and

TABLE II.—Changes in Number of Remote Data Processing Systems

(As of March)

Items	1969	1971	1973	1975	1977
No. of Remote Data Processing Systems	84	213	490	1,190	2,079
Annual Growth Rate (%)	—	53	48	55	39

TABLE III.—Remote Data Processing Application Areas

(As of March)

Items	1972	1973	1974	1975	1976	1977
Finance·Insurance·Securities	24.5% (81)	26.1% (128)	22.7% (174)	18.8% (224)	21.3% (319)	17.6% (365)
Production·Inventory Control and Sales Management	39.1 (129)	38.8 (190)	42.1 (322)	45.5 (541)	43.8 (658)	49.8 (1,036)
Data Processing	11.5 (38)	9.8 (48)	9.1 (70)	8.0 (95)	8.1 (122)	7.4 (153)
Pollution Control	5.5 (18)	7.6 (37)	8.2 (63)	6.7 (80)	7.3 (109)	5.9 (123)
Traffic Control	5.2 (17)	5.9 (29)	5.2 (40)	4.6 (54)	4.1 (61)	3.6 (74)
Commodities Transportation Control	2.4 (8)	2.0 (10)	1.6 (12)	1.3 (16)	1.2 (18)	1.8 (37)
Other Use	11.8 (39)	9.7 (43)	11.1 (85)	15.1 (180)	14.2 (214)	13.9 (291)
Total	100.0 (330)	100.0 (490)	100.0 (766)	100..0 (1,190)	100.0 (1,501)	100.0 (2,079)

Remarks: ( ) number of systems.

converted into second or third generation for totalizing or extension of application areas. Thus, banking type systems are playing a vital role in our economical activities. A nationwide banking system was installed in 1973. This system is a message switching system connected to 7,600 offices of 87 banks. It handles communications pertaining to and settlements of inter-bank domestic exchange transactions. As represented by this example, such systems that cover independent companies and form computer networks tend to increase.

Catching up on computerization efforts

Minor enterprises in Japan are now on their way to computerization. Larger enterprises mostly completed computerization in the early period and have replaced their former systems by on-line systems. In recent years, small business computers and remote information systems have become popular among minor class companies as suitable tools for their business scale. There have also appeared systems that are shared among companies engaged in the same business field.

Such efficient and totalized utilization will become such that small scale business computers, of the stand alone type possessed by minor companies, will be linked to a large scale computer system. Thus, remote information services will play a complementary role as data processing devices for minor class enterprises.

Increase in systems close to our life

The improvement of social and national welfare is being required more and more as Japanese people deem the quality of their lives more important than economical affluence. Therefore, they have begun to pay an attention to systems which satisfy those requirements. NTT has already implemented Emergency Medical Information System, Environment-Pollution Information System, En-route Radar Data Processing System and Foodstuff-Market Information System. Because each pattern is new and different from the others, it is true that there are many problems to be solved in the field of technologies, hardware and software and system operation.

Remote data processing perspectives

Remote data processing perspectives can be viewed from three viewpoints. They are the extension of boundaries, numerical increase and technological development.

Enterprise class systems

Computerization in enterprises and industries has continued growing by nearly 30 percent each year during the past decade. Even now it keeps growing steadily. Remote data processing usage is expected to increase with increasing demand for inter-enterprise type of systems besides the increase of intra-enterprise type of systems. Enterprises tend to implement remote data processing systems that can handle settling accounts, message exchange or business calculation among business related enterprises, a certain group of enterprises or those in the same business.

On the other hand it will be probable that cheaper POS terminals will become more popular among shops or supermarkets. In this case, such systems as are combined with POS terminals will appear. This kind of system will have great impact on society.

With the increase in demand for inter-enterprise systems,

it will be more and more necessary to implement the public data network as a common tool. Utilization of shared type database will also be more frequent among enterprises in the same field or by a certain group of enterprises.

Systems related to social welfare and national life

There are many fields left to be computerized and many problems to be solved. Examples of underdeveloped fields are pollution control, environment protection, medical treatment, education, human and commodities transportation control, disaster countermeasures, commerce and improvement of local life standard. NTT has already researched and implemented such systems as Automated Meteorological Data Acquisition System, Environment-Pollution Information System, Emergency Medical Information System and En-route Radar Data Processing System. Although each of them is obtaining good results, their portion is still very small. Therefore, this class of remote data processing usage today can be said to still be in its infancy. However, considering that many remote data processing systems related to this class are being energetically discussed now, intensive demand will appear before long. (cf. Table IV)

Many systems of this class would take a certain time to become popular with the public. The reason is that the cost and resulting benefits of a system of this class don't balance now or that the present demand is not strong enough. It is generally conceived that governmental subsidization or patronage would be an incentive to the demands of this kind.

Strong similarity exists among individual systems used by local government, in education or in medical treatment. If the cost of system development is reduced far enough and the cost and resulting benefits balance by the standardized implementation or sharing software, this would be good stimulation for the realization of this class of systems.

On the other hand, a system of this class has the aspect that massive data is stored and that it often handles information service or information retrieval services. These characteristics would signify that database oriented systems increase in number. From the standpoint that the systems of this class tend to be used publicly, the security or integrity of data has especially to be assessed carefully.

Personal usage class systems

There are two examples of this class in operation in Japan. One is the system for calculation by pushbutton telephone, called DIALS. The other is the seat reservation system for the "bullet" train using the pushbutton telephone. This service is provided by Japan National Railways. Based on the

TABLE IV.—Examples of Systems Now Being Discussed

Social Needs / Areas	Social Activities	Health and Social Security Maintenance	Social Utility Improvement	Quality of Life Improvement	Intellectual Improvement	System Examples
Medical Treatment	○ ○	○ ○ ○ ○		○ ○ ○	 ○	Emergency Medical Information Regional and Remote Health Care Medical Bibliographic Information Retrieval Hospital Information
Human and Commodities Transportation Control	○ ○ ○ ○ ○	○ ○ ○		 ○ ○		Aviation Control Regional Traffic Control Total Motor Vehecle Control Air Cargo Clearance Port Information
Pollution Control and Environment Protection		○ ○ ○				Air Pollution Control Water Pollution Control Environment Information
Disaster Countermeasures	○ ○	○ ○				Automated Meteorological Data Acquisition Emergency Disaster Information
Education			○	○ ○	○ ○	C A I Bibliographic Information Retrieval
Commerce	○ ○		○	○		Foodstuff Market Information Petroleum Distribution Information
Local Society	○		○	○ ○	○	Home Video Information Municipal Hotel Reservation

number of systems of this class implemented, the present situation is still in its infancy.

To popularize remote data processing for personal use, service providers need to stimulate its demand as a first step. Popularization of this class mostly depends upon whether such service can be provided that is attractive and easy to access at a cheap price.

It is very likely that home computers or on-line terminals will become as popular as telephones or electronic calculaters with the rapid price reduction for microcomputers or minicomputers. In this situation, the second step is expected to be demands for various services appearing from the users. In this stage, remote data processing may be said to be completely popular enough. In the near future, automated telemetering service, reservation service and information service are considered sure to appear. Automated telemetering service is now being put to the test in the suburbs of Tokyo.

Basic technologies

Computer network

As remote data processing service extends its boundaries and increases in number, steady progress in computer network technologies is expected that will enable sharing hardware, software and database. In case of centralized computer systems, the expansion of its size does not necessarily bring economical or functional scale merits. This is considered to be the principal reason why a distributed system has appeared. Besides this, three other reasons can be pointed out. One is that the plan for digital switched data network has appeared from the network technology side. The second one is that the network architecture has been proposed from the computer technology side. The last one is that pertinent device technology has rapidly advanced.

In Japan, NTT plans to implement both a circuit switched network and a packet switched network in early 1979 as a public data network for computer connection. For network architecture, NTT also plans "Data Communication Network Architecture (DCNA)." It defines the interface conditions and protocols among the elements of remote data processing systems. The main aims are the improvement of fiexibility in systems, network transparency, decentralized processing and linkage of independent computer systems. With DCNA, it will be easier to connect computer systems made by different manufacturers.

Remote information services

There are several dozens of companies in Japan that offer remote information services. Usage and technology are expected to change as follows.

(a) The present usage may be summarized as the sharing of computer power at low cost. From now on, the usage level will be more sophisticated, utilizing the highly developed functions of remote information services or applications such as forecast, simulation and so on.
(b) It will be more popular to share and utilize high-quality databases or efficient applications that some other users have developed.
(c) With the rapid popularization of microcomputers and small scale business computers, users will tend to utilize both small scale business computers and remote information services, depending on each merit. They will use small scale ones for simple business data processing or patterned data processing. They will use remote information services mainly for sophisticated and large scale data processing capability.

Database

Although the present utilization of databases is not so popular in Japan, utilization will be prompted more hereafter. The following are expected for database utilization and related technologies.

(a) Enterprise will introduce database systems for business use, because the uniformity of information will be recognized as more important to make business activities more efficient.
(b) Governmental systems often store massive amounts of data. Various merits would be derived from the DBMS adoption. A steady increase is expected on systems in numbers that adopt DBMS if cost-benefit is sufficiently improved, with the technological advance of mass storage devices.
(c) Large scale systems often store bulk information and have nationwide spread. In this case, distributed database will often be preferable when considering avoiding software complexity, reliability improvement, peak traffic smoothing and reduction in communication cost. For this reason, distributed type of database will be popular, especially among large scale governmental systems.
(d) Some remote information service offerers will appear trying to offer specified kinds of databases and applications.

EPILOG

Computer technology was born in the U.S. and assisted greatly in mankind's dream to send people to the moon. This technology was transplanted to Japanese soil carefully nurtured, sprouted and rooted. It also withstood the hardship of the winter caused by the oil crisis and is now growing steadily. Japan has elevated the standard of remote data processing technologies, combined with telecommunication technologies, through persistent efforts and adding various original ideas. Some remote data processing systems may even be said to be unprecedented and to have no similar example in other countries. Remote data processing is expected to have great impacts on society, economy and way of life.

Train operation control system for high-speed railway

by YOSHIRO HAYASHI and SHIGEO YOKOTA
Japanese National Railways
Tokyo, Japan

and

TAIZO NAUCHI
Hitachi, Ltd. Head Office
Tokyo, Japan

INTRODUCTION

It was in 1964 that the Shinkansen made its debut between Tokyo and Shin Osaka. It was extended westward to Okayama in March 1972 and then on to Hakata in March 1975. It is now a more than 1,000-km long traffic artery of Japan and is functioning smoothly.

Constructed as a safety-performance high-speed means of passenger transport en masse in great comfort, the Shinkansen has proven its worthiness, playing a vital role in the economic and social growth of the nation. To have it play its part, the latest technical achievements are introduced to produce its rolling stock, track, electric facilities, etc. and control the operation of these aided by computer. It may be said that the Shinkansen is a typical modern railway operating under a total system.

In this paper, the conditions needed of the train operation control system for a high-speed mass-transport railway and how to meet these conditions will be explained.

DEVELOPMENT OF TRAIN OPERATION CONTROL SYSTEM

The Shinkansen calls for a train operation control system, vastly and basically different from that for conventional railways. One of the conditions attributable to the difference is the incomparable high speed. Here the conventional way of the driver operating his train by keeping his eyes on the wayside signal indications would never work. For this reason we employed the cab signal system together with, what is called, ATC (Automatic Train Control). To meet another condition of mass transport, there is CTC (Centralized Traffic Control) at work, though this is not entirely new. However, when it comes to a high-speed mass transport, like the Shinkansen, CTC has been proved to be an effective means of smooth dispatching and transportation. With the apparatus for these systems and the conventional interlocking device, the Tokaido Shinkansen was opened in 1964.

As the number of trains increased after 10 years, the CTC alone could not help the dispatches owing to a flood of information coming into the control center; therefore, CTC had to be backed up with computer. That is to have the computer do as much as possible to control train operation, collect data, classify these and do the transmission and to have the dispatchers devoting themselves to the job of making judgments by using the computer at will. To meet this demand, COMTRAC (Computer-aided Traffic Control System) was introduced.

OUTSTANDING FEATURES OF TRAIN OPERATION CONTROL SYSTEM

In order to operate high-speed trains in large numbers, safely, precisely and efficiently, the basic requirement for train operation control is to have all the elements of the system work together smoothly under prescribed plans (train diagram and working schedule).

With the trains that run on given rails, unlike any other modes of transport, it is clear that high efficiency can be obtained when they are operated under a meticulously planned diagram, instead of in a haphazard way. Even when confusion takes place for some reason, it is possible for all the elements of the control system to get down to work as a whole with the single purpose of "restoring the prescribed plan." One of the outstanding features of train operation is that it works as all planned out.

When train operation is disrupted seriously by a train accident or a natural disaster, it also is possible under the control system to command all its elements to partially change the plan.

Unlike other industries in general, train operation is a commodity that cannot be kept in stock. It has to be provided to suit the demand of the passengers. Overall revision of train diagram once in two to three years and quarterly minor revisions, therefore, are taken in as a matter of course. The train operation system with these features is shown in Figure 1, divided into three functions of Planning, Doing & Seeing.

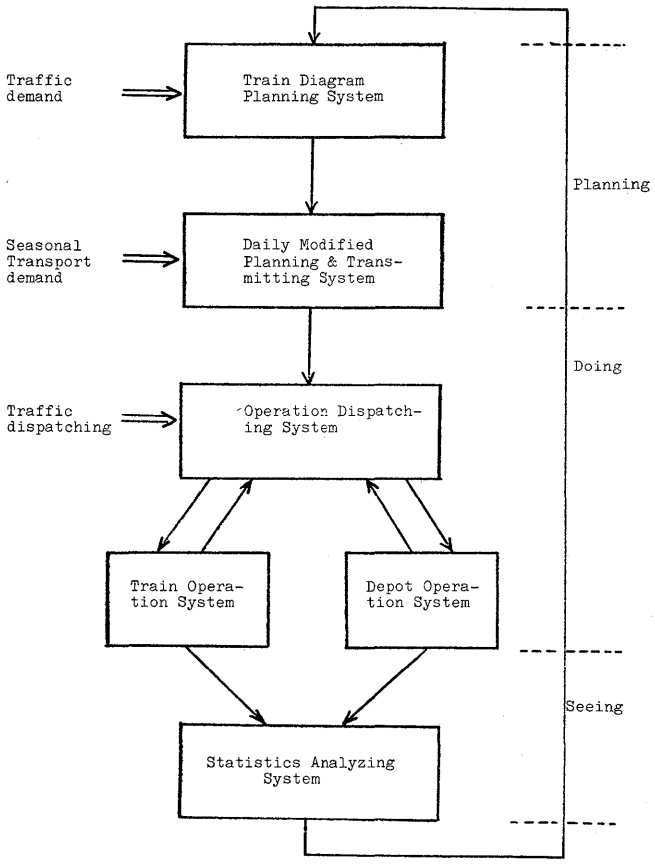


Figure 1—Schematic drawing of train operation control system

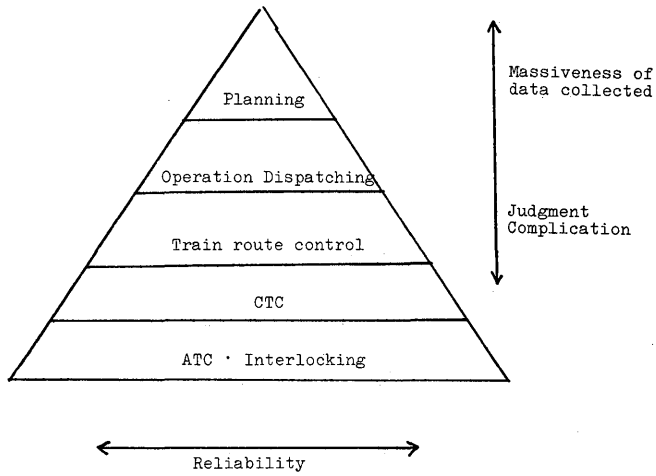


Figure 2—Reliability characteristics in COMTRAC

COMTRAC

Basic concept of reliability design

Various functions shown in Figure 1, closely link themselves together to do the functions. When the reliability, data quality, massiveness of data handled and response time are arranged, we have Figure 2 (COMTRAC Reliability Characteristics). The whole Operation Control System is seen here in a stairway structure, the lower the step, the higher in reliability and the more real time-like. The higher the step, on the contrary, the higher in data accumulation and in judgment complication. Another point to be noted is that the Operation Control System is so made up that once the upper-step function fails out, a lower step fall-back part of the function has failed out.

With this adapted, the stairway structure is taken in the COMTRAC System. As higher reliability and response are demanded of lower functions and as higher-step functions demand more data accumulation and judgment complication, the functions of the system are divided into two groups according to the evaluation criteria. That is, into one group of those functions that strongly demand reliability and responsiveness to be accommodated in the route control system and the other group of functions demanding all sorts of data and others higher-level judgments in the operation adjustment system. As upper level functions call for more and more of those data that the computer is incapable of handling and more and more of human judgment, a man-machine system is needed, leaving the lower-level functions calling for less human intervention to be performed by an automatic system.

COMTRAC functions

A new train traffic control system with computer control is schematically shown in Figure 3. Among many functions necessary for the traffic control, COMTRAC is so designed

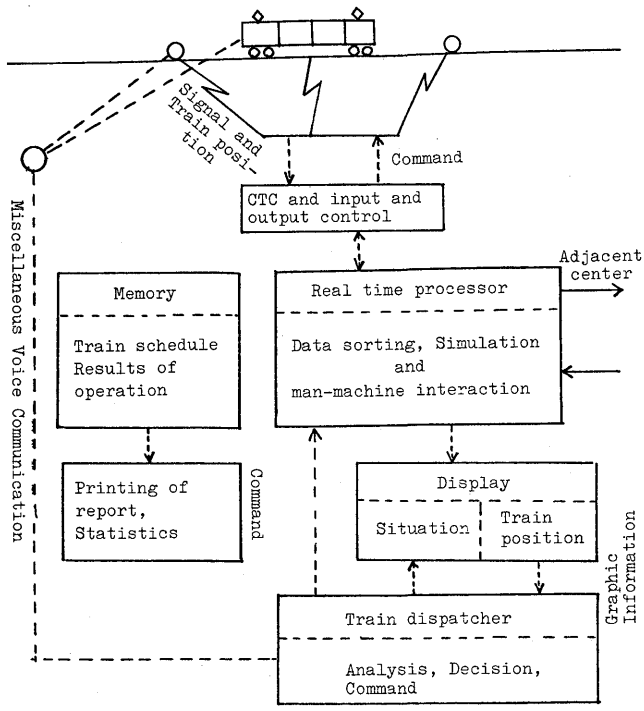


Figure 3—Schematic diagram of COMTRAC system

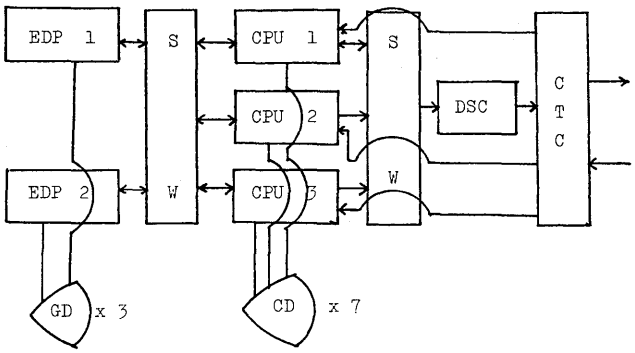


Figure 4—Computer configuration of COMTRAC phase-2

that it can perform the following functions consisted of five subsystems.

1. Planning Subsystem that compiles the plan to implement train and rolling stock turnarounds in accord with the basic and extraordinary plans for train operation.
2. Dispatching Subsystem that monitors whether trains are being operated normally according to the train operation diagram prepared for the day by the Planning Subsystem. On detecting a diagram confusion, the computer automatically renders judgment, if the case is a simple one. If the confusion is serious, due to a train accident, for instance, the train diagram for four to five hours is predicated by simulation and the system prepares a train diagram thereafter and indicates it on GD (Graphic Display). Diagram modifications as decided through the computer are automatically transmitted from time to time to places concerned.
3. Route Control Subsystem that traces each train as diagrammed by the Planning and the Dispatching Subsystems and automatically sets its route. It also monitors the signals in all the stations.
4. The Subsystem to provide data for station announcements. Automation of route control enables the system to detect train positions and train delays, and the information is made use of in making station announcements. The train-to-leave signs on the platforms are automatically controlled, too.
5. Statistics Analyzing Subsystem that prepares all sorts of reports on trains operated and compiles reference materials for administration.

TABLE I.—Types of Passenger Train on Increase (Stopping Station Pattern)

At the time of opening between Tokyo & Shin-Osaka	Between Shin Osaka down to Okayama	Between Okayama down to Hakata
2 types	5 types	9 types

TABLE II.—Route-Setting Automation Checking Growth in the Number of Dispatchers

	Before	Ph-1 when extended to Okayama	Ph-2 when extended to Hakata
A: Train dispatchers	43	52	51
B: Electric railcar train dispatchers	14	15	28
C: Section length (km)	520	680	1060
$\frac{A+B}{C}$	0.110	0.099	0.084

Concept of system composition

In producing a computerized system for the functions explained earlier, the degree of reliability and the response time demanded of each function should be clarified and the unique characteristic of each function need be taken fully into consideration.

The route control system that is directly concerned with train operation is completely a process-control system, setting train routes by the data gathered in the Center by CTC and in accord with the train diagram. It being the last output part of COMTRAC, the highest reliability is demanded of it and the data handled here also must be high in reliability.

The Dispatching System detects diagram confusion by tracing each train with the data provided by the Route Control System and then it really starts displaying its function. As the automatic judgment portion of it is considered to be similar to the monitoring function in the case of train operation, we thought it be better performed under the Route Control System and set one minute as the target for the response time for the diagram simulation function. As for the data transmission function, we thought it hardly matters if acknowledgment of its receipt is made of a change in the plan within several minutes at the worst after the change is decided upon.

The Dispatching System is to be made up as man-machine system as stated before, with man-machine interface centering around GD (Graphic Display) for complicated judgment simulation and around CD (Character Display) for simple item simulation. It is made up as a multicomputer system with two divisions, one a highly reliable system, centering around route control and the other a data processing system centering around train operation adjustment with man-machine equipment included and the fall-back idea partly taken in, as is shown in Figure 4.

TABLE III.—Number of Trains and Lever-Handling Frequencies

	Before	Ph-1 when extended to Okayama	Ph-2 when extended to Hakata
No.of trains a day	170	231	275
Lever-handling frequency a day	4,200	7,000	10,500

TABLE IV.—Availability

	Apr. '76	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.	Jan. '77	Feb.	Mar.	Average
System Availability	100	100	100	99.93	100	100	100	100	100	100	100	100	99.99
Dual Availability	99.89	100	100	99.85	99.96	99.84	100	100	99.90	99.69	99.97	99.92	99.92
No.of system down cases	0	0	0	1	0	0	0	0	0	0	0	0	

(Remark: The one case of system-down was due to software bug)

With this make-up, the Route Control System is as reliable as CTC with the target of 99.99 percent set for the working efficiency, and the Train Dispatching System, also with the target of 99.9 percent, downtime within 10 minutes.

To attain these targets, a 2-out-of-3 computer system was developed for the Route Control System, and a DSC (Dual System Controller) for the Train Operation Control.

As the Train Dispatching System handles a great number of data for many hours, two large-sized computers are introduced for duplex operation.

EFFECTS OF SYSTEM INTRODUCTION & WORKING COEFFICIENCY

In carrying on high-speed mass transport the best way to secure efficiency, safety and preciseness is to perform all the jobs concerned under single command, based on the diagram prepared by the Train Diagram Planning System.

Thus, when a computer system takes care of diagram planning with infallible data, the following effects can be expected:

(1) Infallible train control is possible (Prevention of route missetting)—Prevention of such accidents as train collision to preserve safety is ensured by ATC and interlocking device on the spot. However, missetting of routes for high-speed mass transporting trains gives rise to train diagram confusion; hence missetting is regarded as a serious accident in train operation administration. Although there are no detailed records of route missetting before COMTRAC was introduced, such cases appear to have taken place several times a month. For the large number of various types of trains operating the Shinkansen today, with the stopping station pattern as it is, it surely would be impossible to set routes by human hand without COMTRAC at work, even if the number of dispatchers were increased. After the introduction of COMTRAC, there were three cases of route missetting—two cases, programming misses, and one, an input miss by dispatcher due to man-machine trouble.

(2) Control of trains diversifying in type to meet the demand is possible—Users' demands on the Shinkansen, the main traffic artery in Japan and long in operating section, change from season to season and diverse types of train are operated to meet their demands. Introduction of COMTRAC has thus brought forth a complicated transport form of operating diverse types of train to meet the users' demands. In Table I, the diversification of train types is shown.

(3) The growing number of dispatchers checked by automation of route setting—The number of dispatchers before and after COMTRAC is shown in Table II. Under Ph-1 System, data transmission and rolling stock turnaround control were not performed but the system had the effect of checking the growing number of dispatchers. The systematization also had a notable effect in changing the dispatchers' work from lever handling for route setting to their primary job (train operation adjustment, for instance). With the introduction of Ph-2 System, data transmission and rolling stock turnaround control were performed and the effect of checking the growing number of dispatchers has grown all the more.

(4) It is possible to cope with the demand for high-speed mass transport—The growth of the number of trains and the route setting frequencies are shown in Table III. The route setting by lever operation before COMTRAC averaged once in every one to two minutes. At the time of opening the Shin Osaka-Okayama Section it grew to once in every 30 sec to one minute and then to once in every 10 sec to 20 sec when the Okayama-Hakata Section was opened. It is impossible for man to handle the lever for route setting at such a frequency. The System is presently working nicely with the target reliability value attained. There has been only one case of system-down during the year from April 1976 to March 1977 and this was due to software logic bug. (Refer to Table III.)

FUTURE OF COMTRAC

JNR has two other Shinkansen lines, Tohoku and Joetsu, scheduled for opening within several years and is planning to form a nationwide Shinkansen network. COMTRAC will then be indispensable, as more and more diverse types of high-speed mass transporting trains have to be operated within the network. To be ready for it, COMTRAC will have to expand its scope of control, improve its functions to the infallible extent and grow not only into a train operation control system but also into a traffic control system, as well as to have it linked to MARS (Seat reservation system), Rolling Stock Control System and Management Information System, so that it can provide service more commensurate with the transport demands.

Development of a high-performance universal computing element—PULCE*

by HAJIME IIZUKA and YUTAKA HAYASHI
Electrotechnical Laboratory
Tokyo, Japan

and

KEIKICHI TAMARU and HISASHI HARA
Tokyo Shibaura Electric Co.
Kawasaki, Japan

INTRODUCTION

Since the first introduction of the microprocessor, various kinds of microprocessors have been developed. There have been remarkable achievements both in the level of integration and processing speed. However, most microprocessors developed to date are oriented to device control applications or low class minicomputer replacement. It is true that some bit-sliced microprocessors can operate at very high speeds, but they are neither one-chip, nor do they have much functional capability.

On the other hand, with the current state of the art in semiconductor technology, it seems possible to integrate considerable amounts of computing power in a chip and at the same time to achieve high speed operation comparable to existing discrete processors.

The Electrotechnical Laboratory and Tokyo Shibaura Electric Co. (Toshiba) R&D center are engaged in a research effort in this area and have developed a new high-performance universal computing element—PULCE. This paper describes its architectural design, [1,2] as well as the semiconductor technology develped for its implementation.

BASIC DESIGN

Initial target of semiconductor technology

At the beginning of the project, future trends of semiconductor technology were thoroughly investigated by evaluating pioneering experimental results[3] and the annual rate of improvement in the performance of MOS-LSI's.

The targets for 1977-1978 chip production were fixed in 1972 by the Electrotechnical Laboratory. On the basis of extrapolated chip size, line width in photoengraving technology and annual rate of increase of integration of MOS-LSI's, the number of gates per chip was estimated as more than 5000 but less than 10,000. Assuming an air cooled package, total power consumed in one package should be at most 1 Watt or a little more. Thus power dissipation per gate is limited to less than 0.2mW/gate for integration level of 5000 gates/chip, leaving some room for additional power dissipation in output buffers. An ED-MOS gate in which a depletion type MOS transistor is used as a load was selected as a unit cell for high speed, low supply voltage (5V or less) and high integration density.

Based on preliminary experiments, delay-power product of a MOS gate with fan out of 3 was manipulated to be less than 0.4pJ for a LSI with a line width of 4μm or less. A 4μm line width was assumed in 1972 to be one of the advanced MOS-LSI production standards of 1977. Thus the value of propagation delay time t_{pd} per gate with small interconnection stray capacitance and fan out of 3 was set to be less than 2ns. In a μpu LSI, the loading capacitance of each gate depends on logic design and pattern layout, particularly on wiring length, so t_{pd} of a gate cannot be predetermined. The target value of t_{pd} with fan out of 3 is one of the essential parameters to estimate device design and semiconductor process technology. Overall speed of a μpu LSI largely depends also on its architecture and interconnections.

Basic architectural design

From a standpoint of architectural design, the 5000 gates integration level was one of the two biggest restricting factors, because it was not considered large enough for our design aims of integrating a high-performance microprocessor in one chip.

Another big problem was the number of pins. If we employed a commonly used package with less than 50 pins, we could not achieve a processing capability which would match the internal speed, because interface pins must be

* This work was done as part of the PIPS (Pattern Information Processing System) Project, an ongoing research program funded by the Japanese Government.

used in a time-multiplexed fashion. Furthermore, instruction word-length is shortened and detailed control may be impossible. Therefore, it was decided to adopt a flat package with 80 pins. The development of a high-performance interface to satisfy this condition, as well as the choice of functions to put in PULCE, were the most important design problems.

In view of the technological level of semiconductor and computer systems, and the requirements imposed by the desired applications, we adopted the following considerations as the basis of the architectural design.

Exclusion of sequence control

As LSI μpu technology is based on mass production, its architecture must be general-purpose. Usually, processor functions (instructions) are divided into the following three categories, in which the relationships with general purpose requirements are different.

(1) F type (Functional)—Arithmetic and logical processing functions.
(2) P type (Procedural)—Sequence control functions.
(3) E type (External)—External control functions.

Requirements of the F type instructions do not differ in any applications as far as the microinstruction level is concerned. Accordingly, it is relatively easy to extract functions with wide generality for this type. However, each system entails a different requirement for the P type, especially with respect to interrupt processing, and hence P type standardization is difficult. Furthermore, because of the large number of interface lines and constraints on the amount of hardware, usually only simple functions, which may be unsatisfactory for the purpose of PULCE, can be integrated in an LSI chip. Finally, the E type has special features for each application. But, as these are external functions, only general-purpose considerations and interface expansibility need be dealt with for this type.

In view of the above considerations, and because of constraints on the level of integration and number of pins, we decided not to include a sequence control portion in the LSI chip. Instead we concentrated F type functions. Only the general-purpose requirements and flexibility of interfaces for external implementation of suitable P and E functions are considered.

16-bit basic word length

The length of data processed at one time is very important for this type of μpu. One method is to achieve expansibility by slicing a word into 4 or 8 bits, but there are items, like status information, which cannot always be handled well by the bit slicing, and even 8 bits may be too short for the integration level. Therefore, we have adopted 16 bits as basic word length and 32 bits for interface, and provided PULCE with some architectural features in relation to the simplification of double length data processing.

Emulation oriented

Here emulation means not only interpretation of machine instructions of existing computers, but also includes emulations of new system architectures or intermediate languages directed to high level languages. This is one of the important features of new processors. For this purpose, mask registers and a few mode bits which are effective for extracting fields in a word are provided. Operating in conjunction with a multi-bit shifter, the mask registers serve not only for partial field processing as in (macro)instruction decoding, but also for the usual processing.

Stack function

Pushdown stacks are now employed in various applications and their effectiveness is widely recognized. Therefore, a special hardware support which always keeps the upper portion of a stack in internal registers is provided to increase its processing speed.

Flexible control of hardware details by microprogram

Generally speaking, as the level of a functional module becomes higher, it becomes more difficult to implement or modify a lower level function. In other words, modularity decreases flexibility. This is called "loss of transparency" due to modularization.[4] In the case of such high level modules as PULCE, where general-purpose requirements are important, the loss of transparency must be kept as small as possible. For this purpose microprogram control by unusually long (32 bits) instructions has been adopted for flexible control of hardware details.

Regular structure

It is expected that this kind of μpu will be programmed by general users who are not always hardware specialists. This situation is different from the static microprogramming cases. Therefore, the organization of the instruction repertoire and internal structure should be as regular and as easily understood as possible.

ARCHITECTURE

PULCE is organized in a 3-bus structure around the ALU and a shifter as shown in the block diagram of Figure 1.

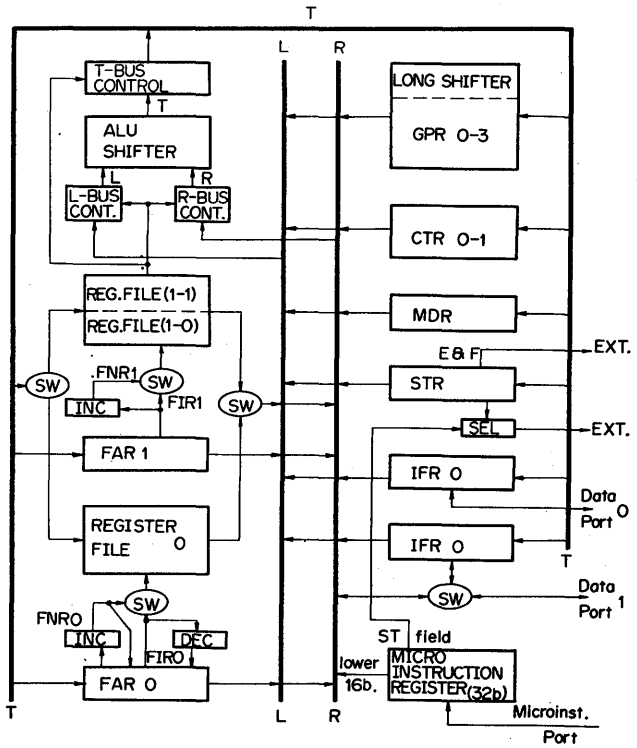


Figure 1—PULCE block diagram (GPR: General purpose register, CTR: Counter, MDR: Mode register, STR: Status register, IFR: Interface register, FAR: File address register, FIR: File indirect register, FNR: File next register, INC: Incrementer, DEC: Decrementer, SEL: Bit selector, SW: Switch)

Functional units

Main registers

The 16 registers listed in Table I are always directly accessible and most of them are assigned dedicated functions as clearly shown by their names. However, FIR's, FNR's and IFR's are provided with some special features described below.

Each FIR is a word in a register file pointed to by FARO-1, so no specific hardware exists for FIR's. Each FNR is a word in a register file whose address is the content of corresponding FAR plus one. Using both FIR and FNR, two continuous words in a register file can be processed without changing the FAR content. This feature is convenient for emulation of system architecture with word lengths of 32 or 64 bits. On the other hand, when SM bit in MDR is set, FNR0 is used for a stack function and its behavior differs from that described above (see *Stack operation*).

IFR's are registers for communications with external hardware attached to PULCE and their input/output connections are made to interface ports via bidirectional buffers. The input/output through these ports are controlled externally instead of by internal microinstructions. Furthermore, IFR0 can be externally set/reset in a bitwise manner, and as a result, this register can be used as an external status register. In addition, if desired, a port of IFR1 can be directly connected to R-bus in PULCE by externally activating a special control terminal. This feature enables a user to either look at data on the R bus or put data on the R bus from interface ports.

TABLE I.—Main Registers

Mnemonic	Name	Length(bits)	Number	hardware*3
GPR	General Purpose	16	4	○
FAR	File Address	4	2	○
MDR	MoDe	16(4)*1	1	○
STR*2	StaTus	16	1	○
FIR	File Indirect	16	2	×
IFR	InterFace	16	2	○
FNR	File Next	16	2	×
CTR	CounTer	16	2	○

Register files

Two register files (RF0 and RF1) of 16 words are provided. RF0 is a general purpose register set, but it could be used for storage of the upper portion of the stack when MDR's SM bit is set (see *stack operation*). The second register file RF1 is divided into two 8 words subfiles, RF1-0 and RF1-1. Seven words in RF1-1 can be used as mask registers.

Arithmetic and Logic unit

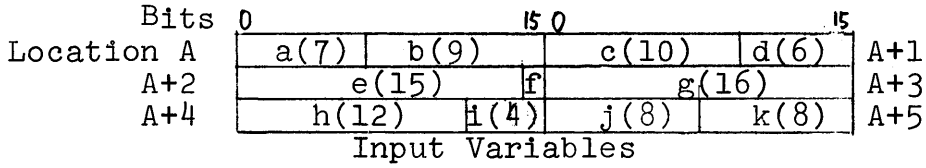
ALU consists of a binary parallel adder, 1-digit decimal adder/subtracter and logic circuits for AND, IOR, XOR, NOR and some special logic functions.

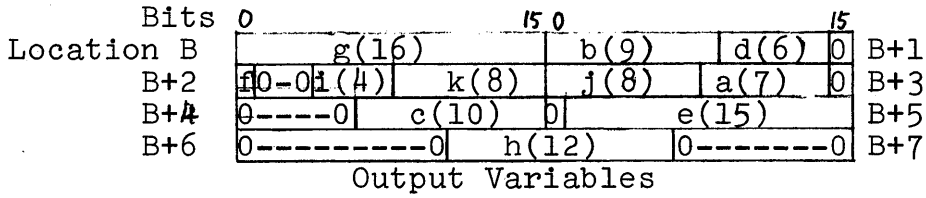
Masking

If designated by the MKfield of a microinstruction, data on L and R buses are ANDed with designated mask registers in RF1-1 before being loaded to ALU. ALU output may also be ANDed with a mask register before being put on the T bus. This feature facilitates effective extraction and processing of a partial field in a register. A trial coding of Church's field processing problems[5] as shown in Figure 2 proved the effectiveness of this feature.

Shifters

PULCE is provided with two types of shifters, a single word length multibit type and a one bit multiword (up to 4 words) shifter. The former can shift a single word 1-15 bits in one machine cycle. Cooperating with mask registers, it

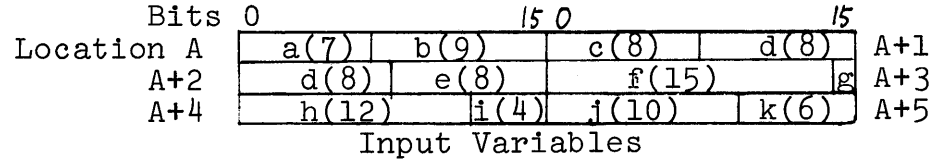


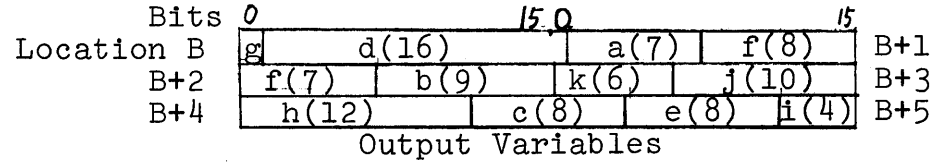


Problem 1: Coding Results

	HP2100	PULCE
Program Steps	86	39
Execution Steps	115	39
memory access	44	14
execution	71	19
mask set	0	6

(a) Problem 1





Problem 2: Coding Results

	HP2100	PULCE
Program Steps	119	46
Execution Steps	174	46
memory access	38	12
execution	136	28
mask set	0	6

(b) Problem 2

Figure 2—Coding results of partial field reformatting

efficiently extracts partial field in a word. General purpose registers GPR0-3 are equipped with a one bit long shifter, and contents of the selected one to four GPR's can be shifted as a one long word. It is effective for multiplication and division.

Stack operation

Since constraints on volume of hardware rule out the use of an independent stack, provision is made for a special hardware support (called Ring Stack) which keeps an upper portion of the stack in register file 0. Thus, when the MDR's SM bit is set, RF0 is accessed by FNR0 in the following manner.

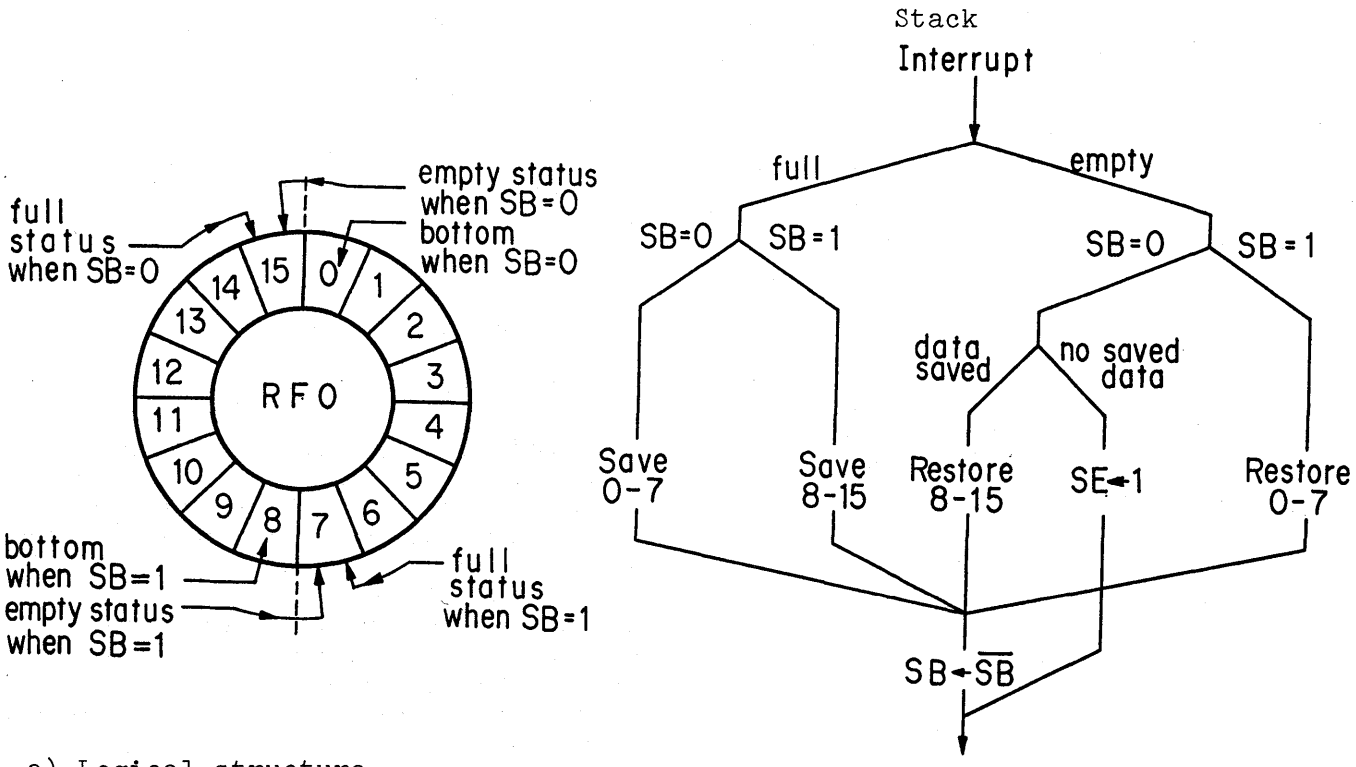
(1) When FNR0 is assigned in the T field of an instruction, FAR0 is incremented at the beginning of its execution. Then at the end of its execution MDR's SE bit is reset, and if FAR0 equals 15 (when MDR's SB is reset) or FAR0 equals 7 (when SB is set) at this time, STR's F bit which indicates "stack full" is set.
(2) When FNR0 is assigned in the R field, FARO is decremented at the end of execution, provided that SE is reset at the beginning. Furthermore, if FAR0 equals 15 (when MDR's SB is rest)/7 (when SB is set), STR's E bit which indicates "stack empty" is also set. However, if SE is set at the beginning, E bit is set and this instruction is ignored.

As is easily seen, (1) and (2) of the above operation correspond to "push" and "pop" operations of the stack. SE is a bit which is set by a microprogram when the stack is completely empty, and by using this bit, stack underflow is detected. SB is used to alternate the set conditions of STR's E and F bits between the middle and the bottom of RF0, essentially making it a ring of registers (see Figure 3a). Accordingly, by saving 8 words in RF0 in external memory when F is set and restoring these 8 words to RF0 when E is set, an upper portion of the stack can always be kept in RF0 (see Figure 3b).

Simulation of Ring Stack has been done and the results are shown in Figure 4. The probability of data transfer to and from external memory is substantially reduced by the Ring Stack, when the number of continuous push/pop operations is less than the hardware stack size (16).

Microinstructions

The microinstruction repertoire is designed not only for ease of application but also for controllability of the μpu. There are six formats shown in Figure 5.



a) Logical structure.

b) Interrupt processing flow.

Figure 3—Ring stack mechanism

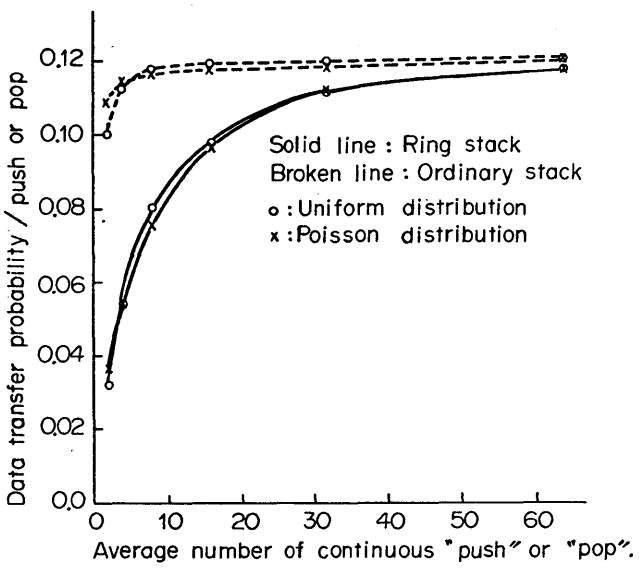


Figure 4—Performance evaluation of the ring stack

RR type instructions

This type is used for operations between registers or a register and 5-bit literal.

Operation code (RR)—RR designates any one of 16 operations consisting of 4 logical operations, 6 binary addition/subtraction, 4 decimal addition/subtraction and two special operations for multiplication and division.

Status assignment (STB, STC)—STB assigns a bit in STR and the next instruction may be ineffective depending on its value. STC bit complements skip condition.

Status save (SS)—SS bit indicates whether status information after execution is saved in STR or not. This control enables the programmer to retain past status information in STR.

Bus control (TC, LC, RC)—TC controls loading of data on the T bus into an assigned register, and RC designates whether the R field is a literal or register number, while LC indicates whether the assigned register content or "0" is put on the L bus.

Counter control (CT)—CT is a field to indicate that one or both of two counters are decremented. It is mainly used for loop control.

RE type instructions

These instructions are for operations between a register and a 16-bit literal. The lower 16 bits (E) of the instruction are interpreted as a literal and put on the R bus. In addition to normal arithmetic and logical operations, RE type is provided with a special operation called pattern match which puts on the T bus an AND of 16 bits information decoded from the lower 4 bits of L input and a literal (E). This is useful for classifying codes, detecting signs in decimal operations, etc.

RH type instructions

These are operations between a register and an 8-bit literal. As this type can take advantage of the masking feature, it is useful for processing of 8-bit data.

SH type instructions

Instructions of this type shift the content of a register assigned in the T field by the amount (maximum of 15 bits) indicated by the R input in one machine cycle. In addition to normal shift modes, there is one special logical shift in which GPR0 is implied as a source register. This is often useful for processing of partial fields in a register, because the original content of the register is not changed by the instruction execution.

LS type instruction

This is a long shift instruction of GPR0-3. Assigning a repeat mode in the MM field, only this instruction can be repeated until the designated counter becomes zero or the skip condition is fulfilled.

NP type instructions

Instructions whose upper 3 bits equal "111" do not have any effect internally. In other words, NP type is designed individually for each system to control sequence and external operations.

SEMICONDUCTOR TECHNOLOGY

n MOS/SOS technology

To actually produce PULCE from the design we have described, devices, that are of high integration density, of low power consumption, of high speed and of small stray capacity had to be developed. Considering the state of the art in semiconductor technology, *n*-channel E/D MOS circuits on sapphire substrate (*n* MOS/SOS technology) seemed to be the most suitable device for this LSI. Especially small stray capacitance associated with n^+ diffusion layer wiring and thin film wiring on an insulating sapphire substrate is suitable for a μpu in which long parallel bus lines meander on the substrate.

Most of the studies of silicon on sapphire technology have been concentrated on CMOS/SOS.[6] There has not been any attempt to fabricate high density *n*-channel MOS logic circuits utilizing SOS technology, but an *n*-channel E/D gate is considered to be superior to a CMOS gate because of its

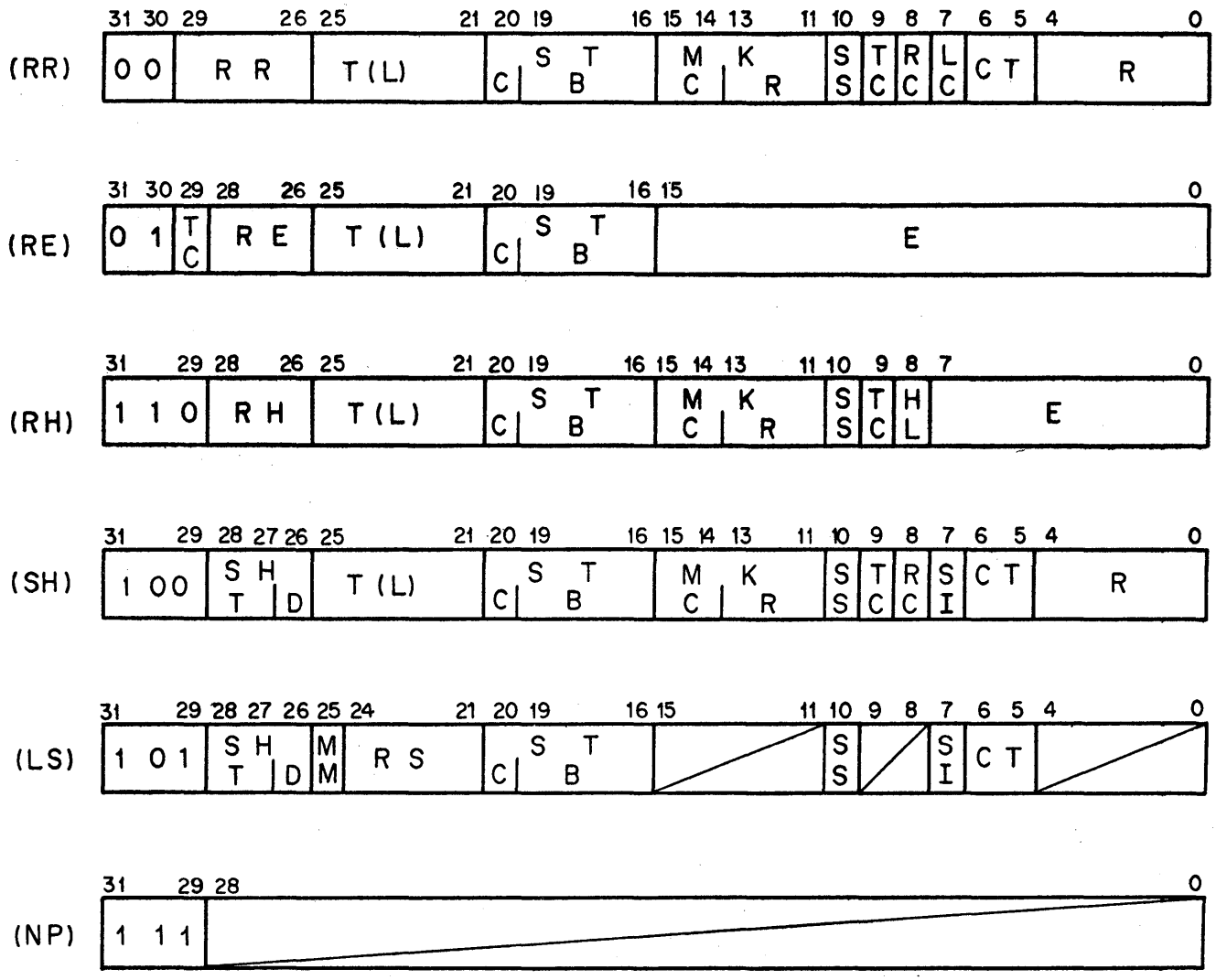


Figure 5—Microinstruction formats (RR: Op. code, T(L): T&L bus, ST: Status, MK: Mask control, SS: Status save, TC: T bus control, RC: R bus control, LC: L bus control, CT: Counter control, R: R bus, RE: Op. code, E: Emit, RH: Op. code, HL: High-low control, SHT: Shift type, SHD: Shift direction, SI: Shift-in control, MM: Multimode, RS: Register select)

high packing density and design simplicity. So, before the final fabrication of the PULCE LSI, three years of R&D were spent developing the high density *n*-channel MOS/SOS process. The method of polishing the sapphire substrate, the hetero-epitaxial thin silicon film growth condition for reduced defect density, the research on origins of drain leakage current and the stabilizing of fine line photoengraving conditions on SOS wafers were included in the R&D project.

Preliminary experiments

Before actual fabrication of the PULCE LSI, functional blocks such as register and arithmetic logical unit RALU and register file RF0 were developed as experimental devices on SOS. The RALU which contains about 2400 gates consists of 16 words by 16 bits local storage, 16 bits adder, two working registers and control circuits. The chip size is 5.00×4.72mm using 6 μm design rule.[7] The typical propagation delay time through 11 NOR gates in carry circuits is 70ns, one third of that on bulk silicon. Register file RF0 consists of 16 words by 16 bits register file, 4 bits file address register FAR0, input buffer circuits and three (T,R,L) busses. This register file performs complex functions such as writing data on the T bus and reading out data on the R and L bus simultaneously. The design rule used is the same 4μm rule as that of the final design. The chip size is 4.41×4.74mm and 1,300 gates are included. The access time is about 30ns or less as shown in Figure 6 and is within 20ns of the design target of 50ns.

The PULCE LSI is fabricated by 4μm design rule and coplanar process. Figure 7 shows the relation of average propagation delay time t_{pd} and power dissipation P_d of the *n* MOS/SOS gate. The parameters of the figure are the ratio of the channel width to the channel length of load transistor of the inverter (4μm:4μm and 4μm:3μm), and the thickness

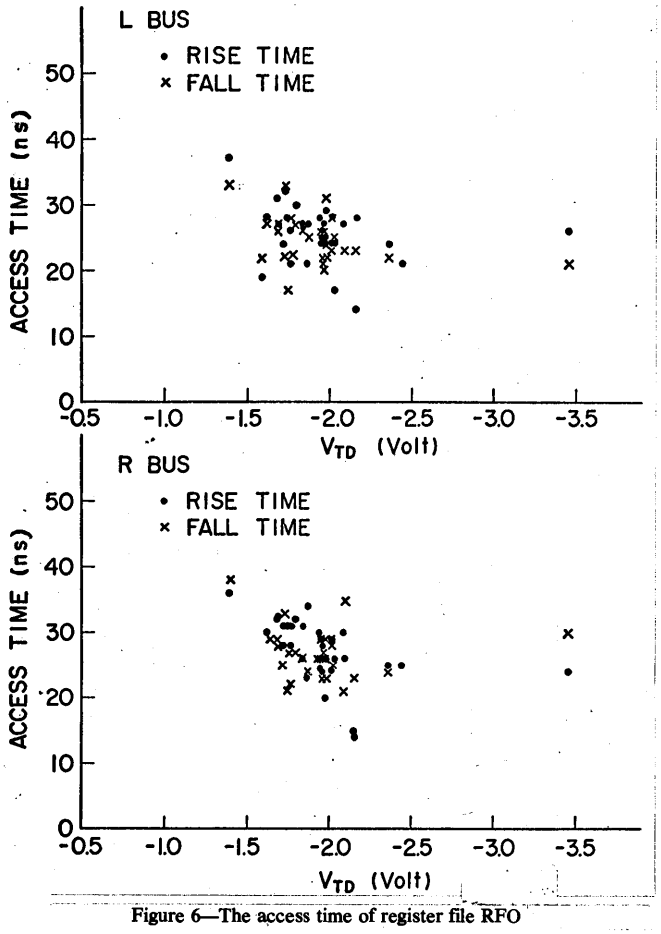


Figure 6—The access time of register file RFO

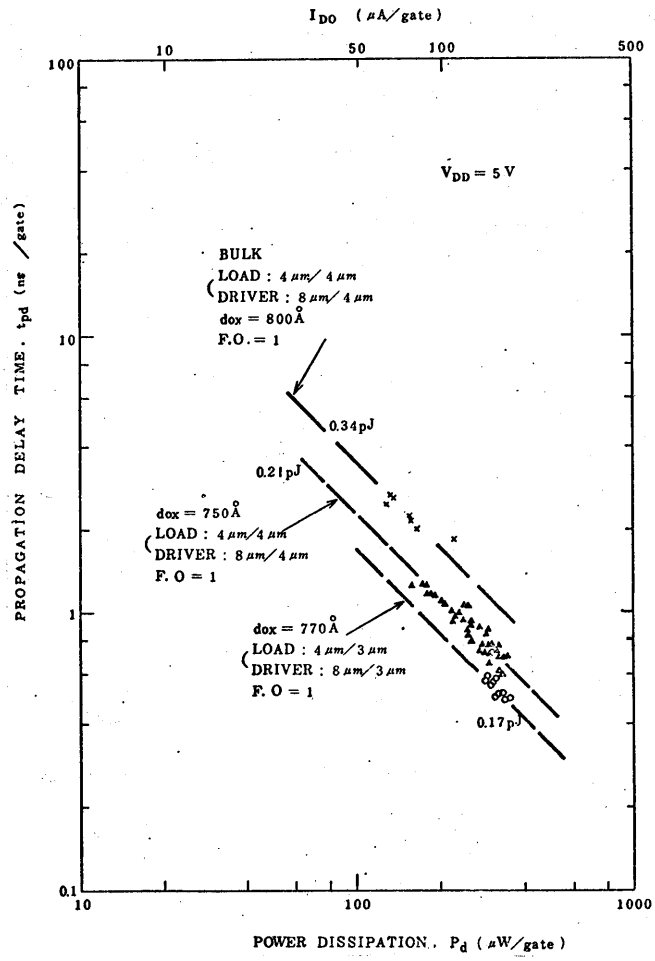


Figure 7—The relation of average propagation delay time vs. power dissipation for n MOS/SOS gates

of gate oxide, 750Å and 770Å respectively. This figure shows the difference of the $t_{pd} \cdot P_d$ product on SOS (0.21pJ) and on bulk silicon (0.34pJ) in the same design pattern. It is clear that SOS is faster than bulk silicon by a factor of 1.6 at the same power dissipation.

Fabrication of the PULCE LSI

For the fabrication of PULCE LSI new technology such as the coplanar process on SOS which improves low gate breakdown voltage and prevents extra drain current in ordinary SOS structure,[8] and circuits with four types of transistors as shown in Table II which reduce power dissipation were used. The gate used E/D circuits, the threshold voltage of which is 0.8±0.2V (V_{TE}) for E type transistors and −2.0±0.3V (V_{TD}) for D type transistors. In addition to these transistors, two other threshold voltage transistors were used. The first one was the load transistor of a memory cell which was fabricated by boron and phosphor ion implantation. The threshold voltage V_{TED} is −0.5V. Power dissipation of memory cells in the register file was reduced by a factor of ten compared to all the same D type transistors. The another threshold voltage transistor was the intrinsic transistor (I type transistor). The pushpull driver using a D type load transistor to drive the large capacitance of bus lines requires power consumption at low output voltage level. Using an E type load transistor causes the voltage to drop to the level of the threshold voltage. To solve this problem, an I type load transistor for which no ion implantation technique was used was incorporated in the pushpull line driver. It has a threshold voltage V_{TI} equal to −0.5~+0.3V. Power dissipation was reduced by a factor of ten and by using a large size load transistor the circuit speed was kept at the same level.

TABLE II.—Four Types of Transistors

Type	Ion implantation		Typical V_T
	Boron	Phosphorous	
E	○	×	0.8V
D	×	○	-2.0V
I	○	×	-0.2V
E-D	×	○	-0.5V

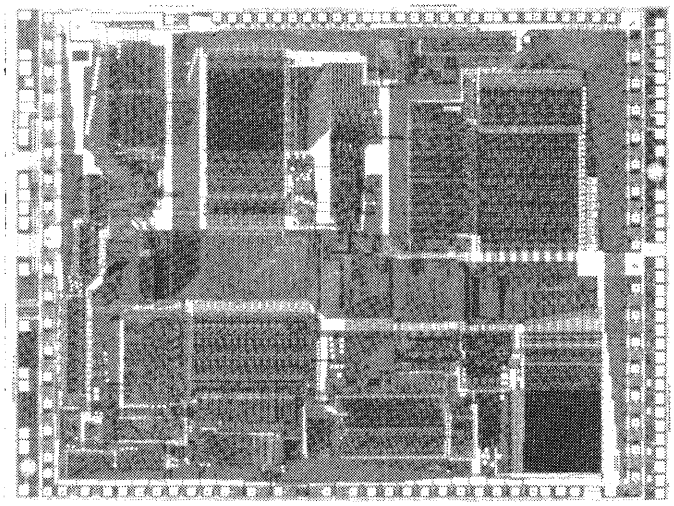


Figure 8—Photomicrograph on the PULCE LSI

Figure 8 shows the photomicrograph of a PULCE LSI. The chip size is 8.85×6.66mm. There are 7,000 logic gates and 20,000 transistors. The processor has a 200ns microinstruction cycle time using a single phase clock. The delay through the look ahead carry path of 10 gates for a 16 bits arithmetic operation is 35ns and the binary addition time including decoder delay ranges from 80ns to 115ns. Multibit shifter delay is 30ns. The chip uses a single 5V power supply and dissipates 1.5W at its full rated speed. Figure 9 is a photograph of the 80 pins package which has five aluminum cooling fins. The heat resistance is 10° C/W at 1m/sec air flow.

TABLE III.—Summary of the Performance of PULCE

Device type	n MOS/SOS
Chip size	8.85 × 6.66mm
Gates in a chip	7000
Transistors in a chip	20000
Package	80-pin flat package with cooling fins.
Power supply	5V
Machine cycle	200ns
Power dissipation	1.5W
Operating temperature	0°C-50°C
Data width	16bits
Microinstruction	32bits supplied from outside
Registers	44
(General purpose)	29
(Mask)	7
(Dedicated)	(16bits) 6
	(4bits) 2
Shifter	0-15bits
(Single word)	1bit
(2,3,4 words)	
Decimal operation	add/sub (1 digit)
Stack	Hardware support
Multiply/divide	No hardware as such, but special hardware available to support these instruction

CONCLUSION

We have described the architecture of a new high-performance microprocessor PULCE and the semiconductor technology developed for its implementation. The first functionally complete LSI PULCE was constructed in the fall of 1977 and is now being evaluated. The performance of LSI PULCE is summarized in Table III.

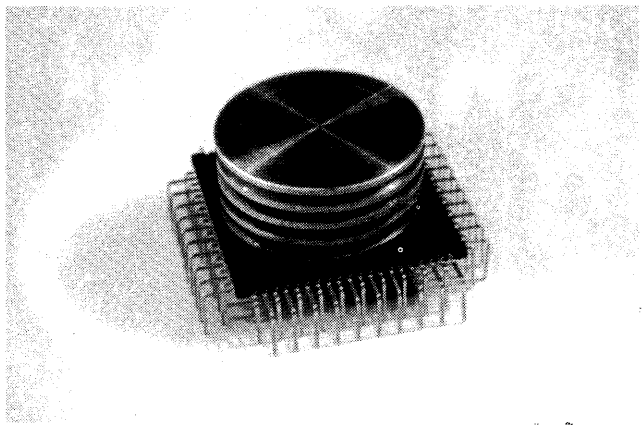


Figure 9—Photograph of 80 pins package

Before the actual implementation of LSI PULCE, we built several experimental machines with PULCE architecture by using TTL SSI's and MSI's, and they have been successfully used to construct processor modules of experimental modular multiprocessor systems.[2,9]

Now we are planning to use LSI PULCE in many research areas, including a data base machine, a LISP machine, a character recognition system and a polyprocessor system. Among these the first application will be a high-performance personal computer, which will be completed by the first quarter of 1978.

ACKNOWLEDGMENT

Many people, indeed too many to mention in this short space, have given the authors support and advice during this project. We wish to express our sincere appreciation to everyone and especially to Messrs. T. Furuya (ETL), H. Tango (Toshiba), T. Sato (Toshiba), and all those associated with this research. The authors also wish to acknowledge the support and encouragement provided by Drs. H. Nisino (ETL), O. Ishii (ETL), K. Kurokawa (ETL), Y. Tarui (ETL; now at the Cooperative Lab., VLSI Technology Research Association) and T. Tarui (Toshiba).

REFERENCES

1. Iizuka, H., and T. Furuya, "Architectural Design of a Microprocessor," *Trans. Inst. Elect. Comm. Eng. Japan,* Vol. 59-D, No. 3, March 1976, pp. 185-195; English Translation appeared in *Systems. Computers. Controls,* Vol. 7, No. 2, March-April 1976, pp. 35-43.
2. Iizuka, H., "Studies on Adaptive Computers," *Researches of the Electrotechnical Laboratory,* No. 767, November 1976 (in Japanese).
3. Tarui, Y., Y. Hayashi, and T. Sekigawa, "Diffusion Self-Aligned Enhance-Depletion MOS-IC (DSA-ED-MOS-IC)," *Proc. 2nd Conf. on Solid State Devices,* Tokyo, 1970; *Supplement J. Jap. Soc. Appl. Phys.,* Vol. 40, 1971, pp. 193-198.
4. Siewiorek, D. P., and M. R. Barbacci, "Some Observations on Modular Design Technology and the Use of Microprogramming," *CMU Report,* July 1974.
5. Church, C. C., "Computer Instruction Repertoire—Time for Change," *AFIPS Conf. Proc.,* Vol. 36, 1970 SJCC, pp. 343-349.
6. Dang, L., P. B. Ashkin, R. Yee, and M. O'Brien, "A CMOS/SOS 16-bit Parallel μCPU," *Digest Tech. Papers,* 1977 ISSCC, pp. 134-135.
7. Iwamura, J., M. Ohhashi, M. Isobe, H. Tango, T. Sato, and I. Yamazaki, "A 2400-gate RALU on SOS," *Proc. 8th Conf. on Solid State Devices,* Tokyo, 1976; *Supplement Jap. Appl. Phys.,* Vol. 16, 1977, pp. 557-560.
8. Tango, H., J. Iwamura, K. Maeguchi, and M. Isobe, "N channel Silicon Gate MOS Devices on Sapphire Substrate," *Proc. 6th Conf. on Solid State Devices,* Tokyo 1974; *Supplement J. Jap. Soc. Appl. Phys.,* Vol. 44, 1975, pp. 225-231.
9. Iizuka, H., O. Ishii, K. Fujii, R. Ohomote, and T. Furuya, "ACE—A New Modular Computer Architecture," *Proc. 2nd US-J Comp. Conf.,* 1975, pp. 36-41.

CACS—Urban traffic control system featuring computer control

by TORU MIKAMI
Nippon Electric Co., Ltd.,
Kawasaki City, Japan

INTRODUCTION

In Japan, the number of automobiles has exceeded thirty million and the number of driver's license holders reached thirty-five million (approximately thirty-five percent of the total population) at the end of 1976. Automobile traffic in many cities has been increasing rapidly year by year. Automobiles and trucks, at present in Japan, perform the most important part of a means for transporting passengers and goods. In fact, for seven years, the amount of transportation by automobiles exceeded that by railways in Japan.

Increased use of automobiles in cities, on the other hand, has caused many serious problems: traffic congestion, traffic accidents, air pollution from exhaust gas etc. For example, more than thirty persons are killed by automobile accidents every day. These problems impose great losses and strains on city dwellers. The city traffic problem has become one of the most important social problems in Japan.

To solve this traffic problem, many means and systems have been considered and applied to the cities. Area traffic signal control systems have been installed in the major cities of Japan. Many big cities have also been equipped with air pollution surveillance systems. Enforcing traffic regulations, extending safety facilities, developing new city transportation systems and many other means have been tried.

It cannot be said, however, that the situation has changed for the better. Both time involved and area covered by traffic congestion in the city are still increasing. As a result even with an increase in the number of vehicles, the transporting capacity of automobiles in the city is decreasing.

In order to solve the city traffic problem, it is basically necessary, in addition to the above, to consider a means for increasing efficient utility of the road network in a city. For this purpose, Comprehensive Automobile Control System (CACS) project, which is introduced in this paper, was established and is continuing. The purpose of the CACS project is to develop an effective means and system for controlling the city automobile traffic comprehensively and efficiently. Technically, this project aims to develop a total information processing system for the automobile traffic by combining car electronic device, car-ground communication, computer control and network technologies. This system will be used in coordination with the existing traffic control system mentioned above.

The CACS project is financed by the Agency of Industrial Science and Technology, Ministry of International Trade and Industry. The project started in 1973 and is planned to continue six years. More than twenty million dollars have been invested up to now. Organizations involved in this project are the Research Association for Comprehensive Automobile Control Technology, Toyota Motor Co., Ltd., Nippon Denso Co., Ltd., Sumitomo Electric Industries, Ltd., Nippon Electric Co., Ltd., Hitachi, Ltd. and others.

SYSTEM OBJECTIVES AND FUNCTIONS

In the CACS project, for analysis of city automobile traffic problems, the following four system objectives have been adopted:

1. Relieve vehicle traffic congestion.
2. Reduce exhaust pollution that accompanies traffic congestion
3. Prevent automobile accidents
4. Increase public and social applicability of automobile use.

In order to achieve these objectives, the following four functions have been set up as technology development target functions:

1. Provide optimal route guidance to moving vehicles while preventing congestion and resultant exhaust pollution during the traffic congestion period
2. Provide travel priority to emergency and public service vehicles
3. Provide drivers with driving information pertinent to safe driving
4. Provide drivers with other advance information on, for example, the state of traffic emergency in a specific area.

In the actual CACS project development phase, these

target functions are realized through the development of five subsystems shown below:

1. Route guidance subsystem—A system which guides drivers to their destinations via optimal routes according to traffic conditions by displaying guidance information visually inside vehicles.
2. Route display board subsystem—A system which gives drivers the information on road congestion status and recommends routes by outside visual displays.
3. Driving information subsystem—A system which brings to drivers attention information on traffic regulations and driving warnings by inside indications.
4. Traffic incident information subsystem—A system which transmits accident information and the like to drivers for the area through which they are driving by car radios enforced to receive concerned signals.
5. Public service vehicle priority subsystem—A system which allows public service vehicles, such as ambulances, fire engines and repair trucks pass through with priority. This system is to operate in conjunction with signal control systems.

These five subsystems constitute Comprehensive Automobile Control System. Figure 1 shows a conceptual configuration of the CAC system. Figure 1 also shows potential relationships in the future between CAC system and representative external systems, such as area traffic signal control system, expressway surveillance system and others. Traffic administration and road administration also have important relations to the CAC system operation.

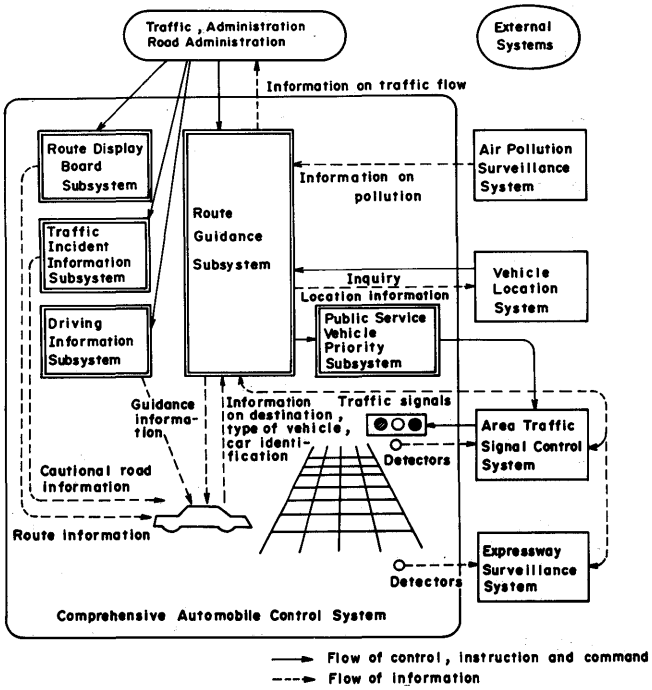


Figure 1—Comprehensive automobile control system concept

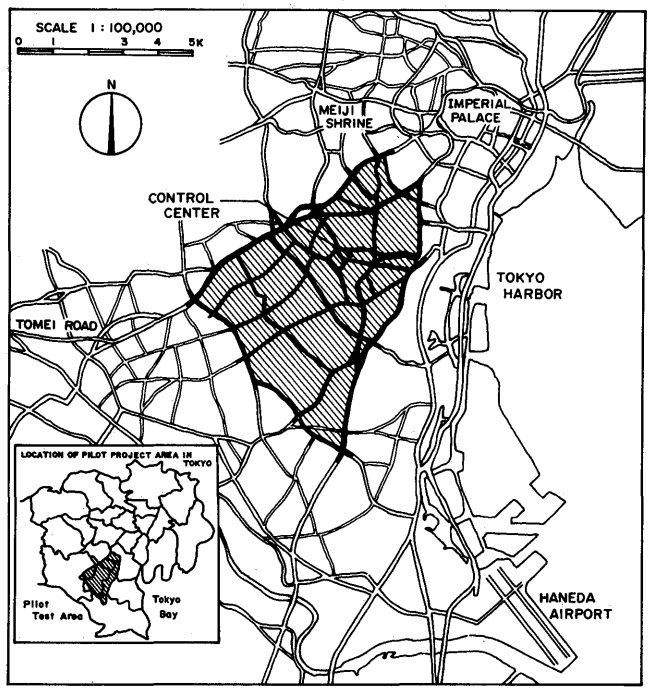


Figure 2—Pilot test area

PILOT SYSTEM TEST

A pilot system test was conducted in the CACS project for a year beginning in October of 1977. Purposes of this pilot system test are to:

1. Evaluate the effectiveness of developed technologies in city streets
2. Collect traffic data for future system design
3. Evaluate social acceptability of this system.

The pilot test area is a 28-square kilometer area in the southwestern section of Tokyo (see Figure 2). This pilot area contains a road network representative of the road network of the entire city; 89 main intersections in this area (including 15 expressway ramps) have been chosen as guidance intersections for route guidance. Three hundred thirty test vehicles, which are equipped with full sets of vehicle units, and 1,000 data collecting vehicles, which are equipped with transmitting parts of vehicle units only, are planned to be used in the pilot test.

PILOT SYSTEM FEATURES

The pilot system is composed of the following five subsystems:

- Route guidance subsystem (RGS)
- Driving information subsystem (DIS)
- Traffic incident information subsystem (TIS)
- Route display board subsystem (RDB)
- Public service vehicle priority subsystem (PVP)

Figure 8 shows the system configuration which realizes functions of these five subsystems.

Main features of the pilot system configuration are as follow:

1. The route guidance subsystem forms the central part of the pilot system. The realtime data on traffic conditions are collected by the route guidance subsystem and are used by other subsystems. The guidance indication is fed to test vehicles only in the pilot area but any point in 23 wards of Tokyo can be designated as the destination.
2. The driving information subsystem mostly shares the same equipment with the route guidance subsystem.
3. The traffic incident information subsystem and route display board subsystem also utilize the equipment, especially computers, of the route guidance subsystem.
4. For the public service vehicle priority subsystem, only the function of identifying public service vehicles and their locations is realized.

ROUTE GUIDANCE SUBSYSTEM

Destination code

A driver who wants route guidance subsystem service determines a destination code number corresponding to his desired destination by looking at the guidance road network map of the pilot system. The destination code is represented by a seven digit number. The first two digits represent a region whose area is approximately equal to that of prefecture (2,000km^2~14,000km^2). A section (whose area is, as in the case of Tokyo, about 30km^2) in the region is represented by the first four digits. Each section is divided into less than seven zones (the fifth digit) and the zone includes less than sixty-three points (destinations) (the last two digits). Each point (destination) corresponds to individual sides of the road between intersections.

Vehicle unit and roadside unit

Each test vehicle is equipped with a vehicle unit. The vehicle unit consists of a destination encoder, display unit, vehicle loop antenna and control unit (see Figure 3). The driver, at the start of his trip, sets the recognized destination code number on the destination encoder. The driver can also select the use of an expressway by pushing an expressway option button. When the vehicle passes over a road loop antenna buried near a roadside unit before entering a major intersection, the destination code number and expressway option information are transmitted to the roadside unit through the vehicle antenna, along with other information, such as vehicle unit number and vehicle class code. A roadside unit is installed near each one of major intersections in the pilot test area. The roadside unit consists of a roadside control unit and a set of road loop antennas laid under the road in each arm of the intersection. Each roadside unit is connected to a computer center with a communication line (see Figures 3 and 5).

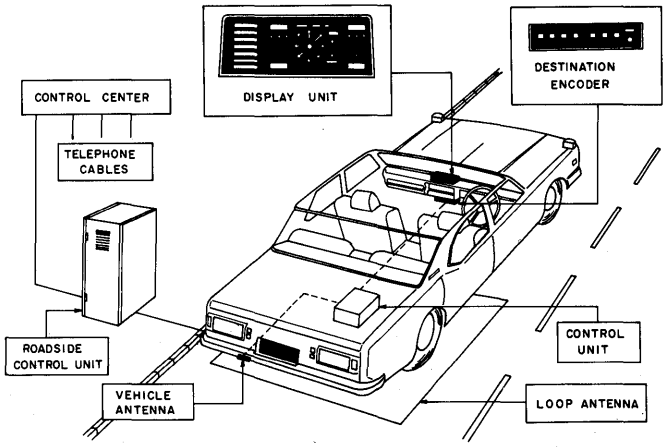


Figure 3—Vehicle unit and roadside unit

Guidance indication

The information from the vehicle unit is fed into the roadside control unit with the information identifying the road from which the vehicle is entering into the intersection. Using the destination code number, expressway option information, vehicle class code and entering link number as key words, the roadside control unit searches a guide table stored in the unit and keys guidance information corresponding to the request. The guidance information is sent back to the vehicle unit through antennas and is indicated on the display unit attached to the dashboard. The indicated information includes the following (see Figure 4):

a. Intersection shape (intersection indicator emblem in Figure 4)
b. Turning direction (indicated with one of seven arrows; the case in Figure 4 indicates a turn to the right)
c. Lane selection before entering intersection
d. Lane selection after passing intersection
e. Slope (UP: uphill, DOWN: downhill)
f. Entrance to expressway
g. Exit from expressway
h. Emergency guidance (detouring instruction)
i. Destination arrival (indicated by flashing arrow).

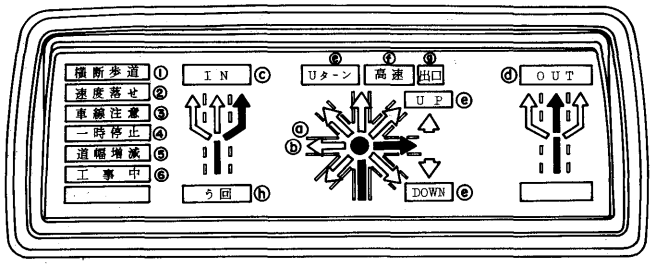


Figure 4—Display unit

Driving in accordance with the guidance indication of this subsystem from the start point to the destination enables the driver to reach the destination in the minimum amount of time even though he is unfamiliar with the place. This guidance indication is automatically updated every fifteen minutes according to changes in present and predicted future traffic conditions.

System configuration

Figure 5 illustrates the route guidance subsystem configuration. One hundred and three roadside units installed at major intersections in the pilot test area are connected to a communication control computer in a control center with 1200 bps speed data communication lines. Six computers are installed in the control center: a communication control computer, central computer, operation control computer, traffic network controller and two route search computers. A traffic network simulater performs a substitute for the route search computers. One hundred and eighty two conventional traffic flow detectors are connected to the system in order to measure the vehicle flow rate and road occupancy.

System operation cycle

General guidance information fed to drivers is automatically updated every fifteen minutes. Every processing required for its renewal, such as traffic data processing, traveling time estimation (arc costs), optimal route computation, guide table formation and so on is consequently carried out in fifteen minute cycles. Some other processings, for example diagnoses of roadside unit efficiency, are also executed every fifteen minutes. The data on traffic flow and road occupancy are collected from detectors at five minute intervals. The information from vehicle units is gathered at one minute intervals. Lengths of these cycles can be changed by commands from an operating console.

The basic operation of this subsystem consists of a sequence of the following three cycle operations. The data on traffic flow, density and running test vehicles are collected in the first fifteen minute cycle. Upon these data, the optimal paths are computed and the guide tables are made in the second fifteen minute cycle. In the third cycle, the guidance for the vehicle is performed according to these guide tables. Therefore, the vehicles are eventually guided on the basis of data for two cycles before.

Information flow and processing

Figure 6 shows an outline of the information flow and processing in this subsystem. The communication control

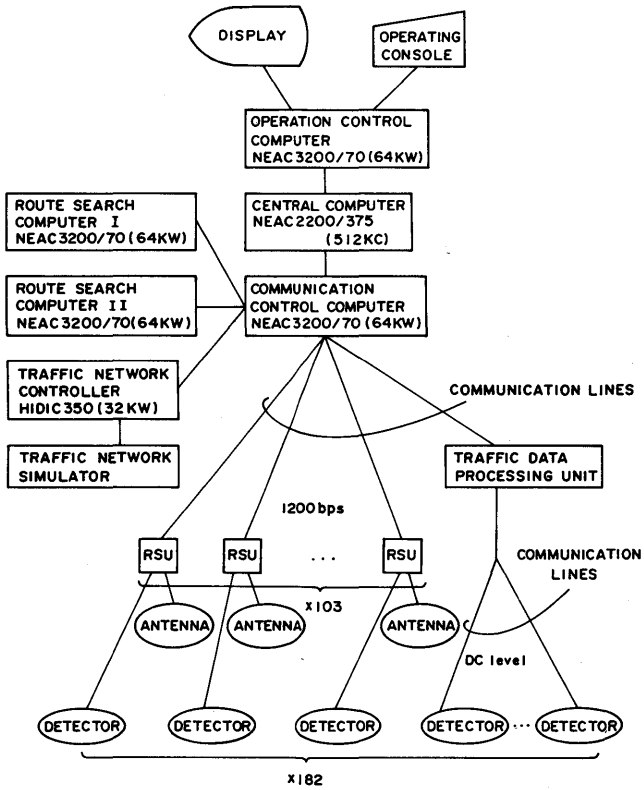


Figure 5—Route guidance subsystem configuration

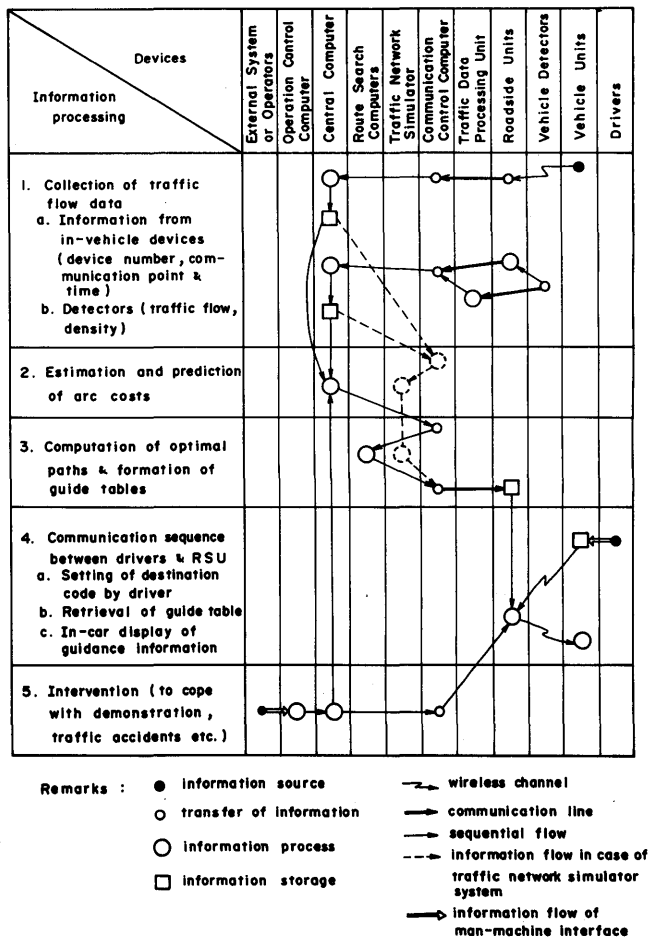


Figure 6—Information flow and processing in route guidance subsystem

computer collects data on points and times of communications between roadside units and equipped vehicles from roadside units and sends the data to the central computer. The data on traffic flow rates and traffic densities, detected by the detectors, are also collected from the roadside units and a traffic data processing unit and sent to the central computer.

In the central computer, an arc cost for each segment of the road network is estimated and predicted on these collected data. Here, the "arc cost" represents the length of time required to traverse the distance from one communication point to the next communication point beyond the intersection. These arc cost data are sent to the route search computers or to the traffic network simulator.

The route search computers compute optimal paths using the arc cost data, pertinent area traffic regulation data and other network data. The computation of optimal route search is performed at each intersection. Time-minimal paths from each intersection (origin) to every point (destination) in the network are computed. For computation economy, the route search network is specially deformed. The network structure is expressed most precisely near the starting point (origin) and its expression becomes less precise as the field goes away from the starting point.

Computation results are condensed into a guide table from every guidance intersection. The guide table is expressed in a combination of list and table structures. The maximum size of the table is less than two kilowords (one word is sixteen bits). The guide table thus formed is transmitted to each roadside unit through the 1200 bps speed data communication line. The use of the guide table at the roadside unit was previously mentioned.

OTHER SUBSYSTEMS

Driving information subsystem

The driving information subsystem gives the driver information helpful to safe driving by inside display and audible signal. When a vehicle equipped with the vehicle unit passes by a buried loop antenna connected to a roadside unit, pertinent area traffic regulations and other information on the road the vehicle is traversing is transmitted to the vehicle unit and any pertinent warning information is shown on the display unit. The warning information is expressed in one of the following messages (see Figure 4):

1. Pedestrian crossing (① in Figure 4)
2. Go slow
3. Change in lane condition
4. Stop
5. Change in width of road
6. Road under construction.

The information on the posted speed transmitted from the roadside unit is stored in the vehicle unit, which checks the vehicle's cruising speed. When the vehicle speed exceeds posted speed limit, an audible warning signal is emitted.

The contents of this driving information, stored in the roadside unit, is updated by the master information in the central file at every system cycle. The contents can also be changed by intervention command from the control room.

Traffic incident information subsystem

The traffic incident information subsystem informs a driver about information on emergencies which affect traffic flow by means of area-limited broadcasting. The broadcast information includes the news on traffic accidents, fire, sudden traffic jam and the like. It also contains advice for the driver such as recommended route, foreseen trip time etc.

In an emergency, the above information, a part of which is made up of arc cost data from the route guidance subsystem, is immediately transmitted from the control center to pertinent TIS roadside units. Each TIS roadside unit broadcasts the information to vehicles through the leaky coaxial cable antennas (LCX). When a vehicle having a car radio equipped with TIS radio adapter enters into LCX range, the vehicle receives the announcements. These announcements interrupt regular radio broadcasts and are received even though the radio is switched off.

Route display board subsystem

The route display board subsystem furnishes data concerning traffic conditions to all drivers of vehicles which are

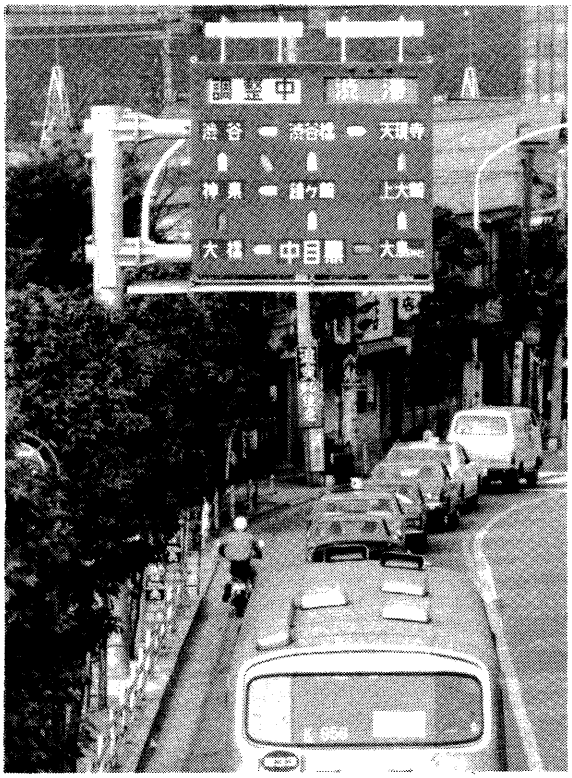


Figure 7—Route display board

not equipped with vehicle units. The information includes data on traffic congestion, accidents and/or recommended routes. This data is indicated on route display boards installed along the road. The names of a number of neighboring intersections and arrows connecting them are illustrated on the board (see Figure 7). The traffic conditions on the route between two intersections connected by an arrow are indicated by the colors of the arrow:

1. Red: heavy congestion
2. Yellow: slight congestion
3. White: no information
4. Green: recommended route to a major intersection

The congestion information and recommended routes result from arc cost data calculated in the route guidance subsystem. They are sent from the central computer system to the route display boards through off-line operator handling.

Public service vehicle priority subsystem

The function of this subsystem is not realized in the present pilot system. Only the capabilities of detecting and tracking the equipped public service vehicles on the central wall type display are checked.

CONTROL ROOM

Figure 8 is a sketch showing total CACS pilot system configuration. A control center is located nearly at the center of the pilot test area. One large-scale computer (NEAC 2200/375), five minicomputers (four NEAC 3200/70 and a HIDIC 350), one special-purpose computer and four specific use processors are installed in the control center.

A control room is included in the control center in order to control and manage operation of the overall system and to supervise pilot test execution. The major devices installed in the control room are a supervisory operating console, four subsystem operating consoles and a wall display (see Figure 9). The wall display includes a central screen display (2.0m×1.5m) with a projector in the rear, color character CRT display, lamp panel and a few indicators.

With the aid of these devices, the following operations are performed in the control room:

1. Initial system set up
2. Starting and stopping the overall operation or partial function of the system
3. System function intervention after an accident, in an emergency situation etc.
4. Recording system operating data
5. Roadside unit status diagnosis
6. Changing system parameters, configuration etc.

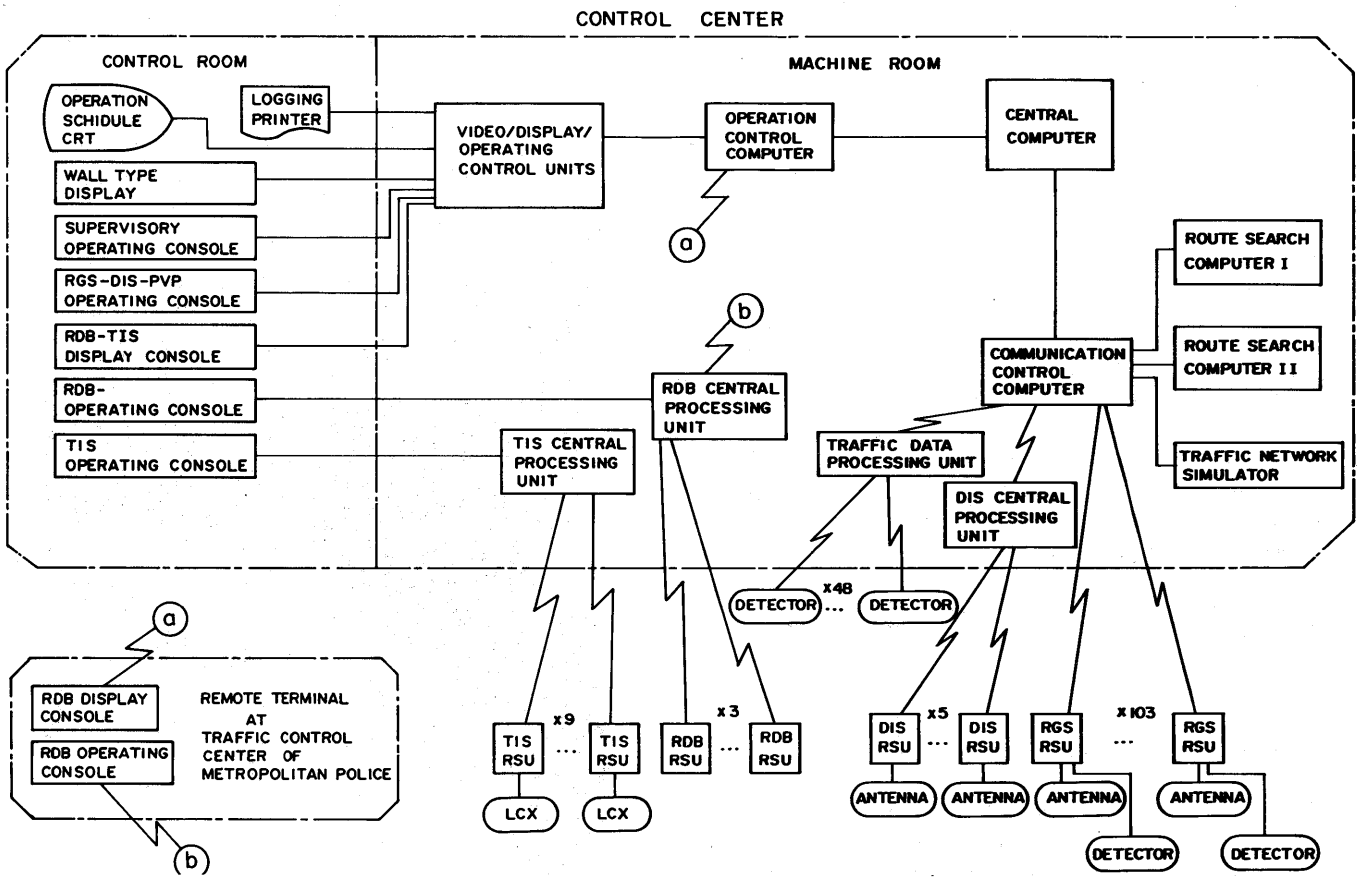


Figure 8—Pilot system configuration

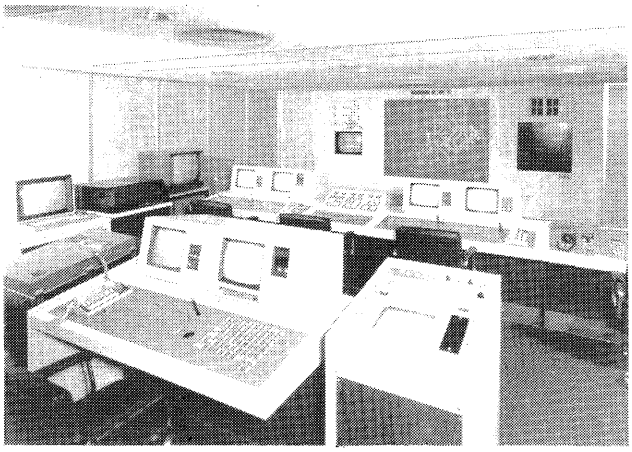


Figure 9—Control room

7. Monitoring traffic conditions
8. Monitoring system and device status

The congestion degree on each road in the pilot test area, public service vehicle location, optimal route and like data are indicated by colors on the map projected on the central screen of the wall display. Information pertinent to tracking a designated vehicle can also be illustrated there. Roadside unit and detectors status data are indicated on the character CRT display. In addition, the computer system's status, test conditions, data and time are shown on the wall display. Most of these figures can also be seen on the color graphic display mounted on the operating console.

CONCLUSION

The Comprehensive Automobile Control System has been described. Essentially, the description has concentrated on its pilot system.

It is expected that the CAC System will produce many beneficial effects on individual drivers and society to:

1. Lighten the driving effort of a driver who is unfamiliar with an area by guiding him on the optimal route to his destination
2. Reduce the time required for traveling from the start point to a specific destination
3. Relieve local traffic congestion in a city
4. Decrease traffic accidents due to careless driving
5. Eliminate the gasoline waste caused by engine idling in traffic congestion
6. Reduce air pollution due to exhaust gas.

ACKNOWLEDGMENT

This paper is presented by permission of the Agency of Industrial Science and Technology, Ministry of International Trade and Industry of Japan. The author is responsible for the facts, opinions and data presented in the paper. The contents do not necessarily reflect the official views or policy of the Agency.

A distributed processing system and its application to industrial control

by Y. MATSUMOTO, O. SASAKI and T. SUMI
Toshiba Electric Co.
Tokyo, Japan

INTRODUCTION

Industrial computer systems are characterized with some of their native features major items of which are:

1. Large amounts of information come from physical equipment, not from human objects.
2. The systems must control physical equipment having no human-like intelligence.
3. Quick response or parallel response is a great concern in most cases.
4. Reliability of computer systems and safety consideration to operators and equipment in case of computer malfunction are vital elements.

More than 20 years have passed since the first installation of computers in process industries. Several steam power plants and some of steel industries were the volunteers in Japan. Table I summarizes the features characterizing the progress of the process control system.

There were some significant milestones during the progress of which the remarkables were the appearances of minicomputers, micro-computers and data high-way.

Technical innovations made after each of these milestones have pushed designers toward the preference for distributed configuration of both system architecture and software.

The attempt to actualize various types of dual, duplex, hierarchical systems using multiple numbers of computers connected through channel controllers or memory bus adapters was motivated by the advent of mini-computers beginning in 1970.

Data high-way is brought into real world as the most useful communication tool to connect computers in medium distance apart. It had the reason of its birth in the process designers' demands that more reliable coupling means without losing economical properties than the one they had in using communication lines were desired. CAMAC data way and IEC bus are also used in subsystems to connect instruments and some kinds of actuating devices.

Micro-computers have been welcomed by process designers in quick rise. The word "distributed control system" used to characterize micro-computer-controlled digital control systems has become popular since 1975. This system has micro-computers in their hearts instead of conventional logic elements.

The functional requirements to the computer systems before 1975 were entirely purposed to control physical objects such as valves and motors. However, since 1976 increasing amounts of jobs to handle those kinds of information concerned with management control and production control have come within the scope of process control system. Fortunately the computers called mega-mini or some others which have main memories sizing more than 256 KB have been brought into market. These have been accepted as the suitable intervening computers between conventional physical control systems and the management information system. In this paper, these kinds of computers are called information control computers.

The latest design on the distributed computer system covering the scope of information control through to physical control and some typical applications to some real industries are described in this paper.

SYSTEM CONFIGURATION

Figure 1 shows fundamental configuration of the distributed processing. Computers (CPU), remote scanners (PI/O) and devices are connected to a data high-way through remote stations. The data high-way adopted here (TOSHIBA TOSWAY 1500 or 6300) is loop type. This type of the data high-way has the following characteristics:

1. Dual lines are applied to have good fault-tolerancy against line failures.
2. M:N communications are available.
3. Twisted paired cable or telephone line may be used. PCM type communication is adopted.
4. Protocol is based on ISO-HDLC. Messages are transparent.
5. Cyclic error checks are made.
6. Speed is 6.3 Mbps for high speed type (TOSWAY 6300) and 1.544 Mbps for low speed type (TOSWAY 1500).
7. Maximum numbers of remote stations are 32 for a loop. Maximum numbers of devices for a station are 16.
8. Maximum length permissive between adjoining remote stations is 4 Km. Maximum total length is 100 Km.

TABLE I.—History of Computer Control Systems

AGE	1958	1963	1965	1967	1970	1973	1975	1977
COMPONENT	LOGGER	COMPUTING LOGGER	PROCESS COMPUTER		MINI COMPUTER	COMPUTER COMMUNICATION LINK	U-PROCESSOR, CROSS CALL DEVICE	MEGA MINI, NETWORK ARCHITECTURE
LOGIC ELEMENT	GERMANIUM TRANSISTOR		SILICON TRANSISTOR	IC	MSI	LSI		GSI
SYSTEM FUNCTION	DATA LOGGING	OPERATOR GUIDE	SET UP CONTROL	DDC	SCC+DDC	PRODUCTION CONTROL	TOTAL AUTOMATION	DISTRIBUTED CONTROL
TOPICS IN SOFTWARE TECHNIQUE	MACHINE LANGUAGE	ASSEMBLER	MULTI PROGRAMMING PROCESS MODEL	DDC AIGORITHM	PROCESS FORTRAN PACKAGED SOFTWARE	FILE SYSTEM	SP, DL, HIPO	HIGHER LEVEL LANGUAGE,
TYPICAL SYSTEM CONFIGURATION	LOGGER / PROCESS	COMPUT-ING LOGGER / PROCESS	PROCESS COMPUTER / CONVENTIO-NAL DRIVE EQUIPMENT / PROCESS	DDC COMPUTER / PROCESS	SCC COMPUTER / DDC COMPUTER / PROCESS	BUSINESS COMPUTER / SCC COMPUTER / DDC COMPUTER / PROCESS	BATCH BC / ON-LINE BC / SCC SCC / DDC DDC / PROCESS	MIS / ON-LINE BC / PC PC PC PC / μCOM μCOM / μ COM / PROCESS

NOTES; SCC(SUPERVISORY CONTROL), DDC(DIRECT DIGITAL CONTROL), PC(PROCESS COMPUTER), BC(BUSINESS COMPUTER)

Computers are connected with data high-way through DMA adapter and controllers. The heart of each remote station is made of micro-processor with 16 bits configuration.

Micro-processors are also used in the remote scanners to make them flexible to requirements variety and intelligence. Another noteworthy application of micro-processors is the one to micro-controllers. The purposes that micro-controllers have are PID (proportional-integral-differential), direct digital control and sequential control of equipment such as motors, valves and actuators. These are used to control positions, speeds and sequential timings. A 16 bit microprocessor (TOSHIBA 40L) is adopted to the micro-controller.

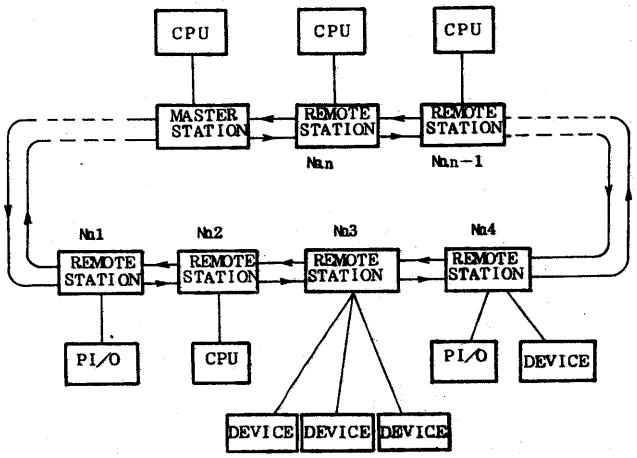
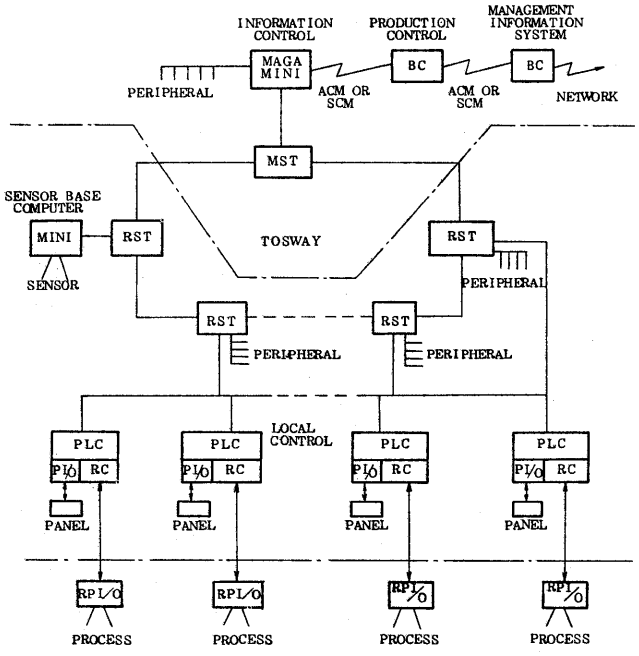


Figure 1—Fundamental system configuration

Figure 2 shows an extended system in which we have one master station (MST) and four remote stations (RST). A mega-mini computer with functions to control management information is connected to the master controller. This information control computer is connected also to business computers (BC) of upper level through communication lines. Four micro-controllers, with the functions of PLC (programmable logic control), RC (remote process I/O control) and PI/O (process input and output), are connected to remote stations. Remote process input and output (RPI/O) devices or remote scanners are controlled by micro-controllers.

The information control computer or mega-mini (TOSHIBA series A70) is a 32 bit computer with 1 Mbytes (addressable max. size is 16 Mbytes) main memory. This computer not only provides advantages that will be obtainable by minicomputer architecture, but also provides capability to process large numbers of data with high precision. This is really suitable to exist in the medium level between



PLC ;PROGRAMMABLE LOGIC CONTROLLER
RC ;REMOTE PI/O CONTROLLER
RPI/O;REMOTE SCANNER

Figure 2—An example of distributed computer system

process control system and production control or management control system.

SOFTWARE

An individual operating system resides in each component computer of a distributed system and each operating system works independently. Application programs to be applied to all the component computers are described with a unique high-level language designed for process control. The language (TPL) is a subset of PL/I plus process control functions.

Specific features are:

- Addition of case statement
- Labeled variable and multiple entrance and eliminated from PL/I, while functions effective for raising the reliability of software, such as argument check, are introduced
- Labeled constants are also accepted for the calculation type data
- Protection for the protected variables is given by declaration
- Debugging at the source program level, such as the tracing of program flow and variables, is facilitated by debugging statements
- Re-entrant function
- Process input/output subroutines
- Task control function.

Although we have several kinds of computers in the distributed system, programmers are freed from learning many kinds of languages.

TPL statements also support the functions necessary to succeed:

- Data transfer between main memories of two computers apart
- "Send" and "receive" data between tasks each of which resides on a different computer
- Data transfer to and from any input and output devices through data high-way.

PERFORMANCE TEST

The software life-cycle for our process control systems are composed of minor elementary cycles including requirements definition, design, implementation, test and maintenance cycle. There are milestones in the cycle of which two are the ones treated in this section.

One of the milestones is intended to validate some selected values of the performance before deciding system architecture and software configuration.

In the other milestone, during test cycle, whole values of performance are measured and tested.

The response time of each task exceeded the required limit is liable to cause damage to the plant equipment and decrease the product quality. In order to avoid these cases occuring both in normal operation and abnormal states, designing optimal arrangement of hardwares and softwares are requisite. The first milestone is intended to validate response time of designed system and to find optimal system structures. Predicting response time by calculation is very difficult. So, digital simulation using SIMSCRIPT is adopted as an automated analysis tool, in this milestone.[1]

Description is made here about an automated analysis tool for rolling mill computer control systems.

Process modelling

The modelling for simulation is shown in Figure 3. A, B, and C, are entities which simulate outside objects commu-

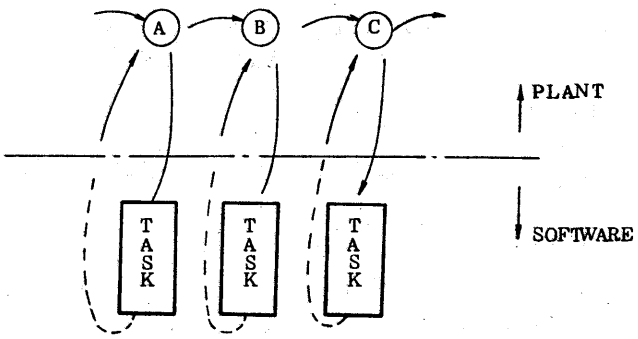


Figure 3—Process model

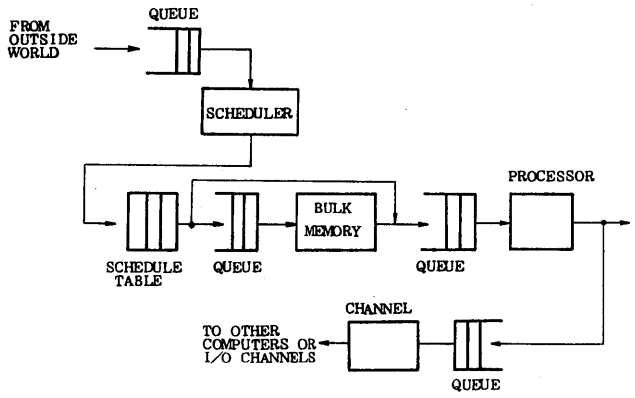


Figure 4—Task procedure model for a computer

nicating with the computer, e.g., the load cell and the hot metal detector.

The modelling for tasks inside of the computer is made as shown in Figure 4. The paths which activated tasks take through the following resources are simulated.

- Scheduling queue
- Bulk to core transfer channel
- Processor
- Peripheral device control

The tasks are terminated after passing through some of these resources.

Input data to the analysis tool

All items which are necessary to model hardware configuration and their abilities, all of task names and their attributes such as average execution times, average frequencies of I/O requests, average sizes of input/output data are inputs to the automated analysis tool.

Average intervals by which interrupts from the outside world occur are also input data to the analyzer.

The analyzer is described with Simscript.

Output data obtained

The following data are obtained as the output of the analyzer.

- Main memory occupancy rate
- Bulk memory transfer channel utilization rate
- Arithmetic unit utilization rate
- System load rate
- Response time of each task
- Execution time of each task
- Bulk transfer time of each task
- Device control time of each task
- Waiting time for execution

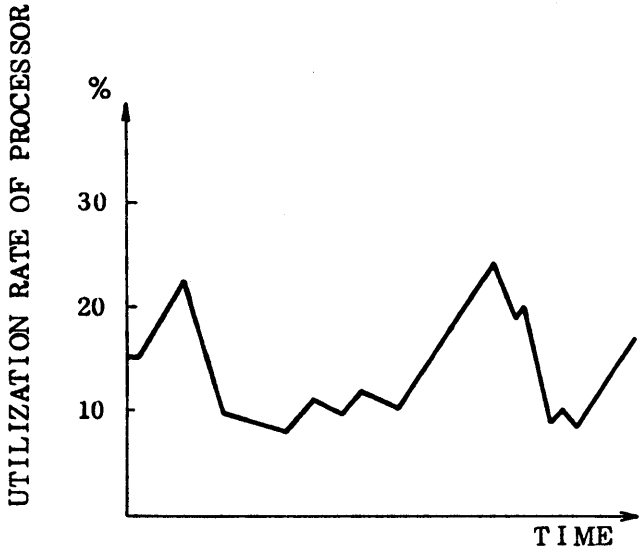


Figure 5—A result of simulation

The maximum and average values of each of these items are obtained.

The result of analysis

An example of results of analysis is shown in Figure 5, Figure 6 and Table II.

In order to obtain the peak load condition in the worst case load, the simulation for some duration of time ranging from 20-25 minutes is required.

According to our experience, a system can be acceptable as having sufficient response if it has an average processor utilization rate below 30 percent and the utilization rates of channels below 70 percent.

Table II shows the accuracy of the analysis tool in the values of response times. The values given by the simulator are compared with the ones measured in the actual operation after the installation. Figures 5 and 6 are the examples of the plotter output.

Utilization rates of the processor corresponding to time are drawn in Figure 5. In the system related with Figure 6, the response time of an activity is required. The activity consisting of six tasks as shown in the figure is activated at time O and terminated at time T. The figure shows that T is the required response time.

TABLE II.—A Comparison of Simulation and Measured Response Time

TASK	AVERAGE OF RESPONSE TIME	
	SIMULATION	ACTUAL
ROLLER TABLE CONTROL	0.27 sec	0.20 sec
MILL CONTROL	0.15 sec	0.20 sec
MILL SET UP	1.01 sec	1.00 sec

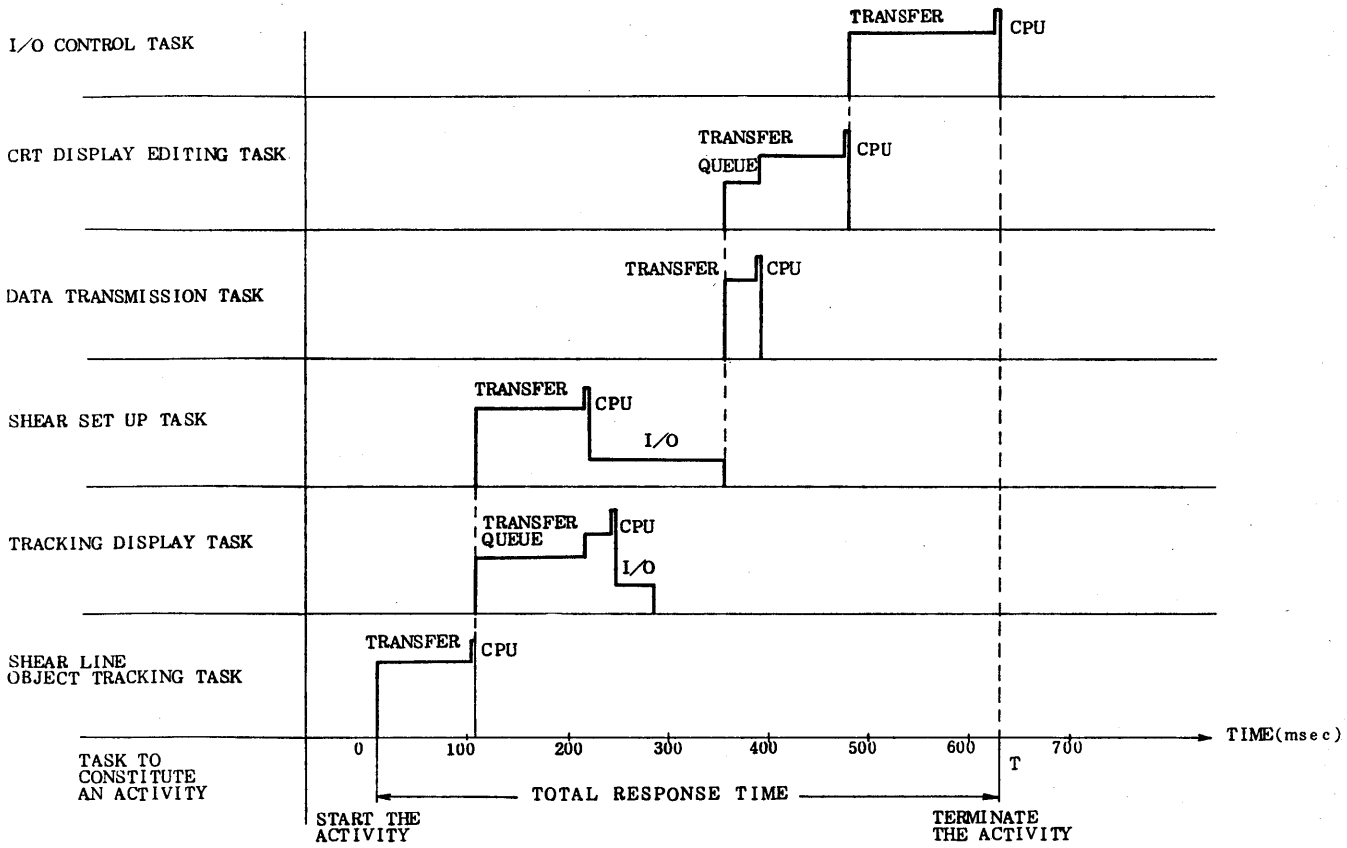


Figure 6—A result of simulation

The second milestone is allocated in the testing cycle. We have in-house computers which are the same types as the computers to be shipped to customers. These in-house computers are connected with a data high-way. Distributed systems to be shipped are able to be simulated with some modifications just for the purpose of test, by this in-house testing facility. We have another computer which we call a dynamic simulator. The programs run in this computer simulate industrial equipment or dynamic behaviors of industrial processes. The dynamic simulator is connected to each one of the above related in-house computers. The programs implemented in the previous cycle are loaded to in-house computers and simulated execution is made.

The structure of the dynamic simulator

The plant processes are classified into continuous processes, batch processes, and dynamic characteristics of individual elements. These are modularized. All kinds of processes can be simulated by combining selected modules.

For continuous process, such a function module that obtains an output from a plural input data through logical or arithmetic procedure is provided.

Elementary modules for the batch process, in case of a rolling mill process for example, simulate roller tables, rolling mills, transfers and shears according to rolling speed, length, width, and rolling directions. It also simulates sensor outputs and the metal detector signals.

How to combine these elementary modules are programmed by using a macrolanguage.

Execution of dynamic simulation

Each of the elementary modules are invoked in the timing generated by the scheduler, and are decoded and executed.

At the execution stage, simulated process data are sent out to the in-house computers, and outputs from the in-house computer are received by the dynamic simulator.

The simulated process data include, for example, metal detector ON/OFF, rolling speed, rolling torque and slab positions.

The outputs from the in-house computers include rolling directions, mill roll gaps, and mill speed set up values. Over 80 percent in number of test items can be validated by this off-site testing facility.

TYPICAL APPLICATIONS[2,3]

From among the currently installed distributed processing systems, some latest examples are picked up with brief

description. The selected fields are:

(1) Iron and steel industry
(2) Water supply and sewage works

Iron and steel industry

This industry in Japan has hitherto pursued high productivity and high yield rate in regard to the products with the targets:

- Enlargement of plant scale with continuity of operation
- Quantitative enlargement of products; diversification of species for products
- Speed up of material handling
- Energy saving.

Success will not be obtained unless we have a total system covering from the management level throughout the works to product planning, process automation and minor physical control.

As an example, herein cited is a large-scale total automation system for a rolling mill. The system shown in Figure 7 is a four-level hierarchy system, comprising the following:

A-level—central on-line system
B-level—mill area on-line system
C-level—process control system
D-level—local controller using a microprocessor or a minicomputer

These subsystems, linked with each other through looped data high-ways, perform high-speed exchange of information.

The process control system is composed of four large-scale process computers with allotted tasks as follows:

No. 1 computer—mill line area,
No. 2 computer—shear line area,
No. 3 computer—refining line area,
No. 4 computer—backup for No. 1, 2, 3.

The peripheral equipment for these computers, totaling 80 units installed at every key point, are controlled through the looped data high-way surrounding the plant.

The process control system, based on the information

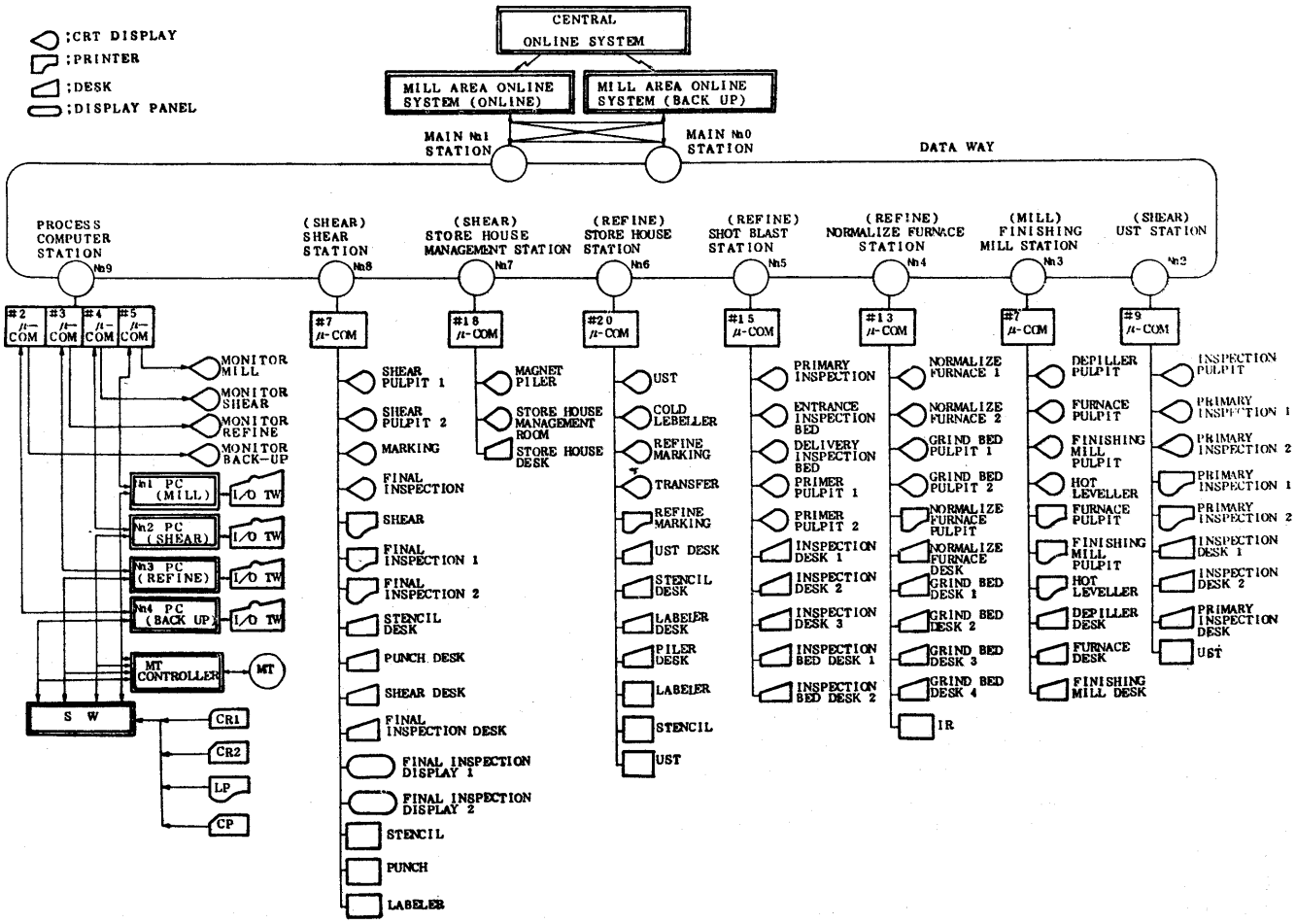


Figure 7—Block diagram of steel mill computer system

exchange with the production control system of the B-level, guides and monitors the total processes covering from the slab yard, which is the entry of the mill line, to the stock yard, which is the exit of the refining line, calculates the optimum setup value, and gives control information to local controllers of the D-level.

The principal functions of the process control system may be summarized as follows:

- Object tracking
- Slab yard mapping
- Furnace temperature control
- Mill Setup
- Cutting schedule calculation and shear setup
- Inspection and classification automation
- Refining automation
- Product-yard mapping
- Shipping schedule
- Automatic handling of objects through total line (roller tables, cranes, transfers and other material handling equipment).

The purpose of introducing this system lies in: improvement of productivity and yield rate, labor saving and improvement of process control accuracy in the range of order receiving to product shipping. This purpose is now being accomplished with excellent results.

Water supply and sewage works

For water supply systems, effective utilization of limited water resources and treatment of waste water have become an important social problem in Japan to be solved quickly.

On the other hand, to meet expanding of area to cover a growing complexity of equipment, increasingly higher techniques of operation are required. Shortage of operators as well as rise of cost of operation are also oppressing the management of projects.

Under such conditions, the introduction of computers is now indispensable to water supply facilities.

Figure 8 shows the total supervisory control system in a latest large-scale sewage works.

The supervision is conducted by the duplicated mega-minicomputer in the central control room, while the control is performed by the distributed systems employing a digital microcontroller or a soft-wired sequence controller linked with each other through the looped data way.

The major functions of this automation are:

Electrical power distribution control
In-house power generation control
Waste-water level control
Sewage pump control
Aeration control
Recirculation sludge control.

The automation for these objects enables the operation of 100 DDC loops and speed control of 20 pumps, etc. This

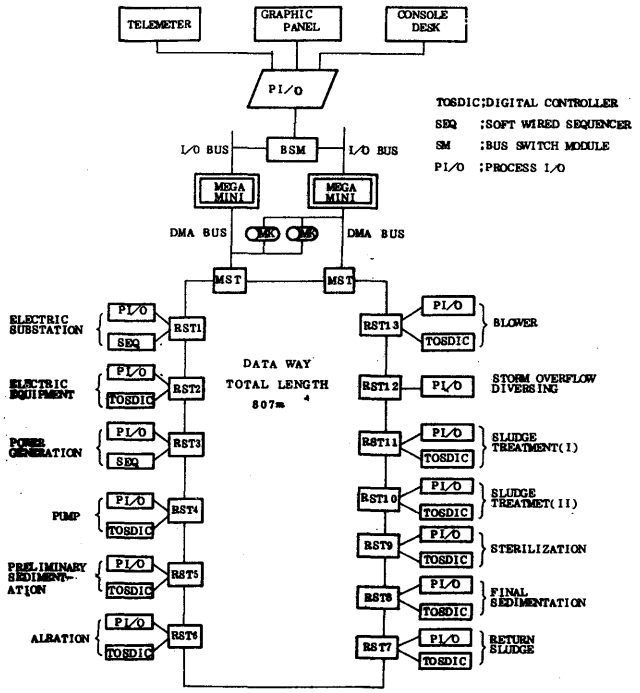


Figure 8—Block diagram of the centralized supervisory control system for a sewage works of Yokohama City

system is making great contribution to the rational operation of the largest sewage works in Japan that treats waste water of one million tons a day.

CONCLUDING REMARKS

There are inherent requirements to process control computer systems that are input and output capabilities both to physical objects and to human objects, restricted response time, safety consideration both to human beings and equipment, and reliability.

It is thought that the distributed system architecture has more capabilities to meet those requirements.

The major components to constitute the system are data high-ways, micro-controllers, mini-computers including mega-mini and remote scanners. These components are more or less modularized in the aspect of both hardware and software. The modularity will increase flexibility of design, quick shipment capability and good maintenance capability.

The concept is believed to be compatible with more complex system requirements in the future.

REFERENCES

1. Matsumoto, Y. and S. Nakajima, "Prediction of Process Computer Performance by Simulation," *Electrical Engineering in JAPAN*, 1973, pp. 128-135.
2. Shimamura, N. and A. Asanuma, "Fully Computerized Wide Flange Beam Mill," *AISE Yearly Proceedings*, 1977.
3. Sumi, T. and O. Sasaki, "Digital Simulation of Grit Chamber Level Control in Sewage Works," *SCSC Yearly Proceedings*, 1974, p. 180.

1978 NATIONAL COMPUTER CONFERENCE STEERING COMMITTEE

Conference Chairman

Stephen W. Miller
SRI International
Menlo Park, CA

Program Co-Chairman

Leonard Y. Liu
IBM Corp.
Poughkeepsie, NY

NCC Committee Liaison

Morton Astrahan
IBM Corp.
San Jose, CA

AFIPS Headquarters Liaison

Gerard Chiffriller
AFIPS
Montvale, NJ

Finance Chairman

Richard Marciano
SRI International
Menlo Park, CA

Keynote Speakers, Co-Chairman

Cuthbert Hurd
Portola Valley, CA

Professional Development Chairman

Gopal Kapur
Danville, CA

Personal Computing Festival—Sessions

Jim C. Warren, Jr.
Redwood City, CA

Operations Chairman

Robert Colten
Gnostic Concepts
Menlo Park, CA

Vice Chairman/Secretary

David Brandin
SRI International
Menlo Park, CA

Program Co-Chairman

Sakti Ghosh
IBM Corp.
San Jose, CA

NCC Committee Liaison

Harvey Garner
Moore School of Electrical Engineering
Philadelphia, PA

Special Activities Chairperson

Ruth Brown
Detroit Transmission Products
Los Angeles, CA

Exhibits Chairman

Ted E. Lorber
Datum, Inc.
Anaheim, CA

Keynote Speakers, Co-Chairman

John Stephens
Excel Minerals Co.
Santa Barbara, CA

Publications Chairman

John Boland
Computer Sciences Corp.
El Segundo, CA

Personal Computing Festival—Contest

Larry Press
Small Systems Group
Santa Monica, CA

Assistant Operations Chairman

Robert Stack
Telefile
Irvine, CA

Past Chairman Advisor

Donald A. Meier
Escondido, CA

1978 NATIONAL COMPUTER CONFERENCE COMMITTEES

TECHNICAL AND PROFESSIONAL PROGRAM COMMITTEE

Co-Chairmen

Leonard Y. Liu
IBM Corporation
Poughkeepsie, NY

Sakti P. Ghosh
IBM Corporation
San Jose, CA

Liz Hoover, Secretary
IBM Corporation
San Jose, CA

Norman Abramson
University of Hawaii
Honolulu, HI

Robert Balzer
Information Science Institute
Marina del Rey, CA

Fred C. Billingsley
NASA Headquarters
Washington, DC

Eric D. Carlson
IBM Corporation
San Jose, CA

Vir Dhaka
Xerox Corporation
El Segundo, CA

Russell H. Dewey
SRI International
Menlo Park, CA

M. Hosaka
University of Tokyo
Tokyo, Japan

David J. Farber
University of Delaware
Newark, DE

Michael A. Harrison
University of California
Berkeley, CA

Lance A. Leventhal
SIMULATION
San Diego, CA

Patrick E. Mantey
IBM Corporation
San Jose, CA

Donald B. Medley
Moorpark College
Moorpark, CA

Alan G. Merten
University of Michigan
Ann Arbor, MI

Richard R. Muntz
University of California
Los Angeles, CA

Susan H. Nycum
Chickering and Gregory
San Francisco, CA

C. V. Ramamoorthy
University of California
Berkeley, CA

Donald J. Reifer
TRW Systems
Redondo Beach, CA

William F. Rousseau
Lawrence Livermore Lab.
Livermore, CA

Jeff F. Rulifson
Xerox Corporation
Palo Alto, CA

William M. van Cleemput
Stanford University
Stanford, CA

Jim C. Warren, Jr.
Dr. Dobbs' Journal of Computer Calisthenics and Orthodontia
Menlo Park, CA

PROGRAM AREA DIRECTORS

Norman Abramson
University of Hawaii
Honolulu, HI

Hideo Aiso
Keio University
Yokohama, Japan

Dennis Allison
Menlo Park, CA

Charles Bachman
Honeywell Information Systems
Billerica, MA

Robert Balzer
USC Information Sciences Institute
Marina del Rey, CA

Fred C. Billingsley
NASA Headquarters
Washington, D.C.

Eugene R. Cacciamani
American Satellite Corp.
Germantown, MD

Vir Dhaka
Xerox Corp.
El Segundo, CA

David J. Farber
University of Delaware
Newark, DE

Peter Freeman
University of California
Irvine, CA

Jack Goldberg
SRI International
Menlo Park, CA

Michael A. Harrison
University of California
Berkeley, CA

William Key
NCR Corp.
Cambridge, OH

Stephen R. Kimbleton
National Bureau of Standards
Washington, D.C.

Stephen J. Kohn
Ernst and Ernst
Cleveland, OH

P. E. Mantey
IBM Corp.
San Jose, CA

John McLeod
SIMULATION
LaJolla, CA

Richard L. Nolan
D.P. Management Corp.
Lexington, MA

Susan Nycum
Chickering and Gregory
San Francisco, CA

C. V. Ramamoorthy
University of California
Berkeley, CA

David C. Rine
Western Illinois University
Macomb, IL

William F. Rousseau
Lawrence Livermore Laboratory
Livermore, CA

J. F. Rulifson
Xerox Corp.
Palo Alto, CA

W. M. vanCleemput
Stanford University
Stanford, CA

Jim C. Warren, Jr.
Redwood City, CA

Donna Williams
Office Systems Integration
Newport Beach, CA

FINANCE

Chairman

Richard A. Marciano
SRI International
Menlo Park, CA

EXHIBITS COMMITTEE

Chairman

Ted Lorber
Datum Inc.
Anaheim, CA

Assistant Chairman—PC

Richard Hernlund, Jr.
Electro Tex/Computer Tex
Houston, TX

Rich Adams
Modcomp
Ft. Lauderdale, FL

Clyde Cornwell
Ampex
El Segundo, CA

Gabriel d'Annunzio
Prime Computer, Inc.
Wellesley Hills, MA

Paul A. Eisner
General Automation
Anaheim, CA

Al Erickson
Dataproducts, Inc.
Woodland Hills, CA

Robert E. Maddy
Tally Corporation
Kent, WA

Connie Magne
Intel Corporation
Sunnyvale, CA

Linda L. Mancini
Verbatim-Information Terminals
Sunnyvale, CA

Lynn McDaniel
Floating Point Systems, Inc.
Beaverton, OR

Jerry Johns
Texas Instruments, Inc
Dallas, TX

Peter M. Johnson
Basic Timesharing Inc.
Sunnyvale, CA

Tom Johnston
Control Data Corporation
Minneapolis, MN

Paul Kraska
Data-100
Minneapolis, MN

Linda A. LaCross
Texas Instruments
Houston, TX

Peter J. Shaw
Megatek Corporation
San Diego, CA

Gene M. Sylvester
MDB Systems, Inc.
Orange, CA

George Wagner
Ball Brothers Research Corporation
St. Paul, MN

Esther Willes
Wangco
Los Angeles, CA

Carl G. Phillipps
Clinton Electronics Corporation
Rockford, IL

Leonard Zaw
Teletype Corporation
Skokie, IL

S. Henry Sacks
Mini-Micro Systems
Hudson, MA

KEYNOTE SPEAKER

Co-Chairmen

Cuthbert Hurd
Portola Valley, CA

John Stephens
Santa Barbara, CA

PROFESSIONAL DEVELOPMENT SERIES COMMITTEE

Chairman

Gopal Kapur
Danville, CA

Richard G. Canning
Canning Publishing Co.
Vista, CA

Fred Gluckson
National Bank of Detroit
Detroit, MI

Ken Garrison
Pacific Mutual Life and Insurance
Newport Beach, CA

PUBLICITY COMMITTEE

Chairman

John Boland
Computer Sciences Corporation
El Segundo, CA

Andrea Graham
Pertec Computer Corporation
Los Angeles, CA

William E. Neel
Computer Sciences Corporation
El Segundo, CA

Ted Lorber
Datum Inc.
Anaheim, CA

Toni Wiseman
Pertec Computer Corporation
Los Angeles, CA

Tom McCusker
Datamation
Los Angeles, CA

Susan Wirtz
Computer Sciences Corporation
El Segundo, CA

PERSONAL COMPUTING FESTIVAL—SESSIONS

Chairman

Jim C. Warren, Jr.
Dr. Dobbs' Journal of Computer Calisthenics
and Orthodontia
Menlo Park, CA

Richard A. Kuzmack
MathTech
Arlington, VA

PERSONAL COMPUTING FESTIVAL—CONTEST

Chairman

Larry Press
Santa Monica, CA

Carl Burlin
Anaheim, CA

Ron Carlson
Marina del Rey, CA

Jim Carlstedt
Santa Monica, CA

Robert Corcoran
Reseda, CA

John Craig
Lompoc, CA

David Llenaresas
San Diego, CA

Brian O'Connell
Woodland Hills, CA

John Perry
San Diego, CA

Neil Rapoport
Sepulveda, CA

Al Sutton
Claremont, CA

OPERATIONS COMMITTEE

Chairman

Robert Colten
Gnostic Concepts, Inc.
Menlo Park, CA

Al Astor
Cybernetics, Inc.
Huntington Beach, CA

Ron Colman
California State University
Fullerton, CA

Ted E. Lorber
Datum Inc.
Anaheim, CA

Demetris Michalopoulos
California State University
Fullerton, CA

Carol Felton
CalComp
Anaheim, CA

Jean-Paul Jacob
Emeryville, CA

Robert Stack
Telefile
Irvine, CA

SPECIAL ACTIVITIES

Chairman

Ruth V. Brown
Detroit Transmission Products
Los Angeles, CA

Tom Jerou
Sperry Univac
Los Angeles, CA

Robert Montgomery
RKO General, KHJ Div.
Los Angeles, CA

Bill Counts
Los Angeles, CA

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

OFFICERS AND BOARD OF DIRECTORS

President

Theodore J. Williams
Purdue University
W. Lafayette, IN

Vice President

Albert S. Hoagland
IBM Corporation
San Jose, CA

Secretary

Sylvia Charp
The School District of Philadelphia
Philadelphia, PA

Treasurer

Walter A. Johnson
Consolidated Papers, Inc.
Wisconsin Rapids, WI

Executive Director

Robert W. Rector
AFIPS
Montvale, NJ

ACM Directors

Herbert Grosch
Sunnyvale, CA

Carl Hammer
Univac Federal Systems
Washington, DC

Stuart Lynn
University of California
Berkeley, CA

IEEE Directors

Samuel Levine
Westport, CT

Stephen S. Yau
Northwestern University
Evanston, IL

Merlin Smith
T. J. Watson Research Center
Yorktown Heights, NY

Data Processing Management Association Directors

Barry D. Lynch
Mercantile National Bank
Dallas, TX

Robert Marrigan
American Fund Raising Services, Inc.
Waltham, MA

Ralph J. Leatherman
Hughes Tool Corporation
Houston, TX

Institute of Internal Auditors Director

William E. Perry
The Institute of Internal Auditors
Orlando, FL

Society for Information Display Director

Carlo P. Crocetti
Rome Air Development Center
Griffis Air Force Base, NY

Association for Educational Data Systems Director

E. Ronald Carruth
Minneapolis School District
St. Paul, MN

American Institute of Certified Public Accountants Director

John Mitchell
American Institute of Certified Public Accountants
New York, NY

American Statistical Association Director

George Minich
World Bank
Washington, DC

Instrument Society of America Director

Arthur Lumb
Proctor & Gamble Co.
Cincinnati, OH

Society for Computer Simulation Director

Per Holst
Foxboro Company
Foxboro, MA

Society for Industrial and Applied Mathematics Director

Donald L. Thomsen, Jr.
SIAM Institute of Mathematics
New Canaan, CT

Special Libraries Association Director

Herbert S. White
Graduate Library, School University
Bloomington, IN

American Institute of Aeronautics and Astronautics Director

Ram K. Khatri
Fairchild Space & Electronics Co.
Germantown, MD

American Society for Information Science Director

Harold Borko
UCLA School of Library Science
Los Angeles, CA

Association for Computational Linguistics Director

A. Hood Roberts
Center for Applied Linguistics
Arlington, VA

NATIONAL COMPUTER CONFERENCE BOARD MEMBERS

Chairman—SCS Representative

Ralph Wheeler
Lockheed Missiles & Space Corp.
Sunnyvale, CA

Vice Chairman—ACM Representative

Smith Dorsey
Fullerton, CA

AFIPS Representatives

Theodore J. Williams
Purdue University
W. Lafayette, IN

Harold Borko
UCLA School of Library Science
Los Angeles, CA

Albert Hoagland
IBM Corp.
San Jose, CA

Treasurer—AFIPS Representative

Walter A. Johnson
Consolidated Papers, Inc.
Wisconsin Rapids, WI

Secretary
IEEE—CS Representative

Lowell Amdahl
Northridge, CA

DPMA Representative

Floyd Harris
Life Insurance Company of Georgia
Atlanta, GA

ACM Representative

Smith Dorsey
Rockwell International
Anaheim, CA

NATIONAL COMPUTER CONFERENCE COMMITTEE

Chairman

Russell K. Brown
Moore Paper Company, Inc.
Houston, TX

Harvey L. Garner
Stanford University
Stanford, CA

Morton M. Astrahan
IBM Research Laboratory
San Jose, CA

Jerry Koory
HW Systems
Van Nuys, CA

Secretary

Al Hawkes
Sargent & Lundy Engineers
Chicago, IL

Irwin J. Sitkin
Aetna Life & Casualty
Hartford, CT

Jeffery D. Stein
On-Line Business Systems, Inc.
San Francisco, CA

Portia Isaacson
Richardson, TX

1978 NATIONAL COMPUTER CONFERENCE

Chairman

Stephen W. Miller
SRI International
Menlo Park, CA

1979 NATIONAL COMPUTER CONFERENCE

Chairman

Merlin Smith
T. J. Watson Research Center
Yorktown Heights, NY

SESSION CHAIRMEN

Amarel, S.
Antal, J.
Austing, R. H.

Balzer, R.
Basili, V.
Bell, C. G.
Berezin, E.
Billingsley, F. C.
Birmingham, D. J.
Buzen, J. P.

Cash, J.
Chandler, J. P.
Chang, J.
Cotton, I. W.
Couger, J. D.
Crandell, G. M., Jr.

Denning, P.
Dewey, R. H.
Dhaka, V. A.
Dodd, G. G.
Dunaway, D.

Edwards, G. C.
Estrin, T.

Faggin, F.
Farber, D. J.
Feng, T.
Finch, F.
Frederick, T. J.
Fu, K. S.
Fuller, S. H.

Gaines, R. S.
Gates, G.
George, J.
Gerhart, S.
Glaser, G.
Goupillaud, P. L.
Green, C.
Gwyn, C. W.

Hart, P.
Helmers, C. T., Jr.
Holzman, D. L.
Huskey, H. D.

Inada, N.

Jensen, E. D.

Karplus, W. J.
Keplinger, M. S.
Kimbleton, S. R.
Kiviat, P. J.
Kling, R.
Kobayashi, H.
Kohn, S. J.
Korsan, R.

Lauer, D. T.
Leeman, G.
Lefkovitz, H. C.
Levitt, K. N.
Lien, D. A.
Lissandrello, G. J.
Little, L. J.
Lodahl, T. M.
Lum, V. Y.
Lyon, J.

Madrigal, O. S.
Magnuson, W., Jr.
Manning, E.
McKinney, J.
McLeod, J.
McMurtry, G. J.
Medley, D. B.
Merten, A.
Mikami, T.
Miller, E. F.
Moore, G.
Mulder, M.

Navathe, S.
Nolan, R. L.
Nutt, G. J.
Nyborg, P. S.

Oliver, P.

Parker, D. B.

Quann, J. J.

Rainey, L. J.
Reddy, R.
Roberts, L. G.
Rousseau, W. F.
Sammet, J. E.
Shriver, B.
Siewiorek, D. E.
Smoot, O. R.
Standish, T.

Taylor, R. W.
Termin, L. M.
Towsen, J. F.
Tucker, C.
Turn, R.

vanCleemput, W. M.
Vick, C. R.

Wagner, G. E.
Warren, J. C., Jr.
Watkins, S. W.
Wessler, B.
Wilson, W.
Witherspoon, J.

Yau, S. S.
Yourdon, E.

Zimmer, R.

PARTICIPANTS

Almon, W.
Amano, M.
Anderson, R.
Appleton, D.
Armer, P,
Ashenhurst, R. L.
Atwood, D. W., Jr.
Aweida, J.

Barbacci, M.
Baumgartner, T. H.
Bennett, J.
Bennett, J. L.
Borgerson, B. R.
Brandel, R.
Brown, R. R.

Calderin, M.
Campos, I.
Capon, P. C.
Carter, W. C.
Case, R. P.
Chai, D.
Chattergy, R.
Chemerys, L.
Chiu, W. W.
Chooljian, S.
Cochran, A.
Conway, D. L.
Couperus, J.
Crocker, S.
Cundiff, W. E.
Curran, M.

Derman, I. H.
Devlin, J. P.
Dickinson, R.
Doddington, G. R.
Donnelly, G.
Dunn, D. A.
Dunning, C.

Eckhouse, D.
Eddington, J. T.
Eger, J. M.
Emery, J.

Faggin, F.
Frenzel, L. E.
Fuller, S. H.

Gambrel, J.
Garrison, K. T.
Giammo, T. P.
Gilberg, D.
Gilmore, J. T.
Goetz, M. R.
Goodman, J.
Green, W. B.
Griswold, R. E.
Guetzkow, H.
Guillen-Williams, R. N.

Haggard, A.
Hampikian, R.
Hanson, M. C.
Hanson, M. L.
Hartley, J. N.
Hartley, P. A.
Haselman, L. C.
Hastings, T. N.
Heikes, R.
Herre, H.
Herscher, M. B.
Herzog, B.
Hill, R.
Hoffmeier, J.
Hogan, L.
Hopper, G. M.
Hord, R. M.
Horning, J. J.
House, P. W.
Howard, J.
Howden, W. E.
Hsiao, D.
Hubbard, S.

Ibbett, R. N.
Ibuki, K.
Ingalls, D.

Jain, A.
Jeffrey, S.
Jelinek, F.
Jensen, E. D.
Jobs, S.
Johnson, R. S.
Jones, A.

Kapur, G. K.
Kane, J.
Kennedy, J.
Kildall, G.
Koplow, H. S.
Korn, G. A.
Kotok, A.

LaBelle, C. D.
Lambert, E.
Larson, H.
Lauer, D. T.
Lavington, S. H.
Lea, W. A.
Lee, D.
Lee, J.
Leeman, G.
Leet, G.
Leet, M.
Lehmer, D. H.
Lincoln, N.
List, B. H.
Lloyd, R. H. F.
Lowry, S.

Machover, C.
Magraw, D. B.
Mangione, P. A.
Manning, E.
Martin, T. B.
Mason, R. O.
Matlin, G. L.
McMohan, F.
Melkanoff, M. A.
Miller, A. R.
Mills, D.
Mills, H. D.
Morris, C. J.
Muran, T. L.

Nash, M. S.
Nelson, D. L.
Nelson, T.
Nesbitt, D. E.
Newport, C. B.
Nimtz, R. O.
Nolan, J.
Norwood, F. W.

O'Leary, D. L.
Olson, D.
Operbeck, H.
Orchard, G.
Orren, J. L.
Osborne, A., Jr.

Padegs, A.
Peddle, C.
Perkins, H.
Perlis, A.
Peters, L.
Petritz, R.
Plumly, L.
Pollock, K.
Press, B.
Puthuff, S.

Ramm, M.
Ray, C. D.
Reddy, R.
Roark, M. L.
Roe, G.
Ruskin, V. W.
Russell, R.
Rybczynski, T.

Sayward, F.
Schaefer, D.
Schkolnick, M.
Schoman, K., Jr.
Schroeder, M. D.
Schubert, R.
Segal, R.
Shemer, J.
Sherman, R. H.
Shetler, T.
Siewiorek, D. E.
Sitkin, I.
Smith, R. E.
Spitzer, R.
Sridharan, N. S.
Stieger, W. H.
Sussman, S. S.
Svigals, J.
Swatik, D. S.

Tellefsen, J.
Thayer, P. B.
Thompson, D.
Thorensen, R.
Tonies, C.
Tropp, H.

Valek, R.
Waldinger, R.
Walker, P. M.
Ward, M.
Ware, W. H.
Wecker, S.
Weiss, E. A.
Welch, J.
White, D. E.
Whittier, R.
Williams, P.
Wilson, H. B.
Wohl, A.

Yao, S. B.

Zachman, J. A.
Zaphiropoulos, R.
Ziehe, T. W.

NCC 78 REFEREES

Abbey, Duane C.
Agrawala, Ashok K.
Aiken, Robert M.
Aiso, Hideo
Archibald, J. A., Jr.
Austing, Richard H.

Balakrishnan, A. V.
Barbosa, Lineau
Barnes, Bruce H.
Bassler, R. A.
Batson, Alan
Bennett, John L.
Berk, Toby
Bernstein, Philip A.
Bise, Robert G.
Blomgren, George H.
Bork, Alfred
Brackett, John W.
Bruell, Steve
Burns, William J.
Burton, William D.
Buscher, David J.

Carey, Bernard
Chandrasekaran, B.
Cheyellena, B. F.
Clema, Joe K.
Cooprider, Lee
Crenshaw, Edsal G.
Culpepper, L. M.

Dalphin, John F.
Daniels, Walter E.
Danner, Lee
Davidson, Donald
Davis, Alan
Davis, Carl G.
Day, William H. E.
Dewdney, A. K.
Dixon, Louis F.
Dunaway, Donna K.
Duncan, Karen A.
Dutton, Ronald D.

Eccles, William J.
Elliott, Glenn R.
Estrin, Thelma

Firschello, Oscar
Fly, William W.
Fong, Elizabeth
Fox, Phillip W.
Frank, W. L.
Frederick, Terry J.
Friedman, Lee A.
Fu, K. S.
Fujino, Kiichi
Futrelle, Robert P.

Gammon, W. Howard
Gaudot, Frank J.
Goldhirsh, I.
Gonzalez, Mario
Goto, Eiichi
Graham, Susan
Gucker, George

Hagiwara, Hiroshi
Hakozaki, Katsuya
Hamblen, John W.
Hanna, William E., Jr.
Hansen, John C.
Harris, Floyd O.
Hatta, Hiroshi
Hattori, Mitsuhiro
Hoffman, Robert H.
Holmes, Harvard
Hord, Michael R.
Hosaka, Mamoru
Howard, John
Hwang, K.

Ibuki, Kimio
Iizuka, Hajime
Inada, Nobukazu
Ishida, Haruhisa
Ishii, Osamu

Jensen, Alton P.
Jordan, Harry F.

Kahn, Kevin
Kfoury, Denis J.
Kornfield, N. R.
Kubo, Hidehito

Lamothe, Ray J.
Lampson, Butler
Lauer, Donald T.
Lazowska, Edward
Leeman, George
Levin, Roy
Levine, Leon
Ligler, George
Liskov, Barbara H.
Long, Harvey S.
Lukas, George
Lum, Vincent Y.

Machover, Carl
Madrigal, Orlando S.
Madron, Beverly B.
Maekawa, Mamoru
Managaki, Masao
Mantey, John
Maskewitz, Betty F.
Mathews, Walter M.
Matsushita, Yutaka
McClain, William J.
McMurtry, George J.
Mead, Robert
Michunas, M. D.
Mikami, Toru
Miller, E. F., Jr.
Mills, Wayne
Miura, Takeo
Modesitt, K. L.
Moll, Robert N.
Morton, A. Kent
Moto-oka, Tohru
Murakami, Kunio
Muraoka, Youichi
Naemura, Kenji
Neumann, Peter G.
Neurath, Peter W.
Nishino, Hiroji
Nishizaki, Minoru

Odaka, Toshihiko
Ohno, Yutaka
Ohsuga, Setsuo

Patil, Suhas
Patterson, David A.
Perry, James M.
Pfleeger, Charles
Pizer, Stephen M.
Potts, Jackie
Powell, John E.

Quann, John J.

Richardson, Debra
Rosin, Robert F.
Roth, R. Waldo
Rotolo, Louis S.

Sagawa, Shunichi
Saito, Nobuo
Sands, J. E.
Sekino, Warren T.
Shelly, Gary B.
Shimamura, Kazuya
Shorecliffe, E. H.
Shriver, Bruce
Siegel, H. J.

Smith, Burton
Smith, Eugene B.
Spaniol, Roland D.
Stokes, Gordon E.
Stranart, J. C.
Suga, Masakazu
Sugano, Takuo
Sugimoto, Masakatsu
Sugiura, Nobunori

Takashima, Kensuke
Takei, Kinji
Tamaru, Keikichi
Tarui, Yasuo
Taulbee, Orrin E.
Tausner, Miriam R.
Taylor, Robert W.
Teichrow, David
Toda, Iwao
Tokoro, Mario
Tomaru, Keisuke
Tucker, Edwin K.

Uchida, Shunichi

Weinberger, G.
Weiss, Stephen F.
Whinston, Andrew B.
Williams, Robin

Yamada, Hiroshi
Yamamoto, Masahiro
Yanayama, Hiroshi
Yarbrouga, L. D.
Yasnoff, William
Young, J. W.
Yovits, M. C.

Zislis, Paul M.

AUTHOR INDEX